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

2019

№ 1

Теми випуску:

- ***Теоретичні та методологічні основи програмування***
- ***Інструментальні засоби та середовища програмування***
- ***Моделі та засоби систем баз даних і знань***
- ***Математичне моделювання об'єктів та процесів***
- ***Прикладні засоби програмування та програмне забезпечення***

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 № 1 від 14.02.2019 р.

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

Підписано до друку 05.03.2019. 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 2019

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 2019

Founded in March, 1999

CONTENTS

УДК 004.42:510.69 https://doi.org/10.15407/pp2019.01.003

М.С. Нікітченко, О.С. Шкільняк, С.С. Шкільняк, Т.А. Мамедов

ПРОПОЗИЦІЙНІ ЛОГІКИ ЧАСТКОВИХ ПРЕДИКАТІВ З КОМПОЗИЦІЄЮ ПРЕДИКАТНОГО ДОПОВНЕННЯ

Досліджено нові програмно-орієнтовані логічні формалізми – логіки часткових предикатів з операцією (композицією) предикатного доповненням, названі LC. Подібні операції використовуються в різних варіантах логік Флойда – Хоара з частковими перед- та після-умовами. В даній роботі вивчаються LC пропозиційного рівня – PLC. Описано пропозиційні композиційні алгебри та мови PLC, запропоновано та досліджено відношення неспростовнісного логічного наслідку за умов невизначеності. На цій основі для PLC однозначних предикатів побудовано числення секвенційного типу.

Ключові слова: логіка, предикат, логічний наслідок, секвенційне числення.

Вступ

Розроблено багато різноманітних логічних систем, які успішно застосовуються в інформатиці й програмуванні [1]. Проте класична логіка має принципові обмеження, які істотно ускладнюють її використання. В основу цих систем зазвичай покладена класична логіка предикатів та базовані на ній спеціальні логіки (модальні, темпоральні, програмні тощо). Обмеження класичної логіки роблять актуальною проблему побудови нових, програмно-орієнтованих логічних формалізмів. В останні роки запропоновано різні підходи до побудови логік, орієнтованих на потреби моделювання, специфікації та верифікації програмних систем [2 – 5]. Перспективним напрямком побудови програмно-орієнтованих логік є спільний для логіки й програмування композиційно-номінативний підхід. Логіки, збудовані на його основі, названо композиційно-номінативними (КНЛ). Різноманітні класи КНЛ описано та досліджено в [6 – 10].

В даній роботі досліджено нові програмно-орієнтовані логічні формалізми – логіки часткових квазіарних предикатів з предикатним доповненням. Такі логіки названі LC. Характерною особливістю LC є наявність спеціальної немонотонної операції (композиції) предикатного доповнення. Подібні операції використовуються в різних варіантах логік Флойда – Хоара з частковими перед- та після-умовами [11]. Основна увага в роботі зосереджена на вивченні LC пропозиційного рівня – PLC. Досліджено властивості композиції предикатного доповнення. Описано пропозиційні композиційні алгебри та мови PLC. Для LC однозначних предикатів запропоновано відношення неспростовнісного логічного наслідку за умов невизначеності. Досліджено властивості цього відношення, на цій семантичній основі для PLC однозначних предикатів побудовано числення секвенційного типу.

Поняття, які тут не визначаються, тлумачимо в сенсі [7, 9]. Для полегшення читання нагадаємо основні визначення.

V-A-квазіарний предикат – це часткова неоднозначна, взагалі кажучи, функція вигляду $P : {}^{V}A \to \{T, F\}$. Тут $\{T, F\}$ – множина істиннісних значень, ${}^{V}A$ – множина всіх *V-A*-іменних множин, тобто множина всіх однозначних функцій вигляду $d : V \to A$. Трактуємо V як множину предметних імен (змінних), A – як множину базових значень.

Множину всіх значень, які неоднозначний предикат P може приймати на аргументі (даному) d, будемо позначати $P[d]$.

1. Квазіарні предикати реляційного типу

Далі будемо розглядати неоднозначні (недетерміновані) предикати реляційного типу, або *R*-предикати.

Клас *V-A*-квазіарних *R*-предикатів тут позначаємо Pr^{V-A}.

Кожний *V-A*-квазіарний *R*-предикат $P : {}^{V}A \to \{T, F\}$ на даному $d \in {}^{V}A$ може приймати лише значення T, лише значення F,

обидва значення T та F, а також може бути невизначеним. Тому для R-предиката P множина $P[d]$ може бути однією з множин $\varnothing$, $\{T\}$, $\{F\}$, $\{T, F\}$.

Кожний R-предикат P можна однозначно задати за допомогою 2-х множин:

– область істинності, або T-область

$$T(P) = \{d \in {}^{V}A \mid T \in P[d]\};$$

– область хибності, або F-область

$$F(P) = \{d \in {}^{V}A \mid F \in P[d]\}.$$

Для R-предикатів область невизначеності, або $\perp$-область, цілком залежить від T-області та F-області. $\perp$-область $\perp(P)$ R-предиката P визначається $T(P)$ та $F(P)$ так:

$$\perp(P) = \overline{T(P) \cup F(P)} = \overline{T(P)} \cap \overline{F(P)}.$$

V-A-квазіарний R-предикат P:

– частковий однозначний або P-предикат, якщо $T(P) \cap F(P) = \varnothing$;

– тотальний, або T-предикат, якщо $T(P) \cup F(P) = {}^{V}A$; тоді маємо $\perp(P) = \varnothing$.

Для P-предикатів пишемо $Q(d)\downarrow$, якщо значення $Q(d)$ визначене, та пишемо $Q(d)\uparrow$, якщо значення $Q(d)$ невизначене.

Таким чином, для P-предикатів T-область та F-область можна визначити так:

$$T(Q) = \{d \in {}^{V}A \mid Q(d)\downarrow = T\},$$

$$F(Q) = \{d \in {}^{V}A \mid Q(d)\downarrow = F\}.$$

Тоді $\perp(Q) = \{d \in {}^{V}A \mid Q(d)\uparrow\}$.

Клас V-A-квазіарних P-предикатів будемо позначати PrP^{V-A}.

Використовуючи області істинності та хибності, можна виділити [9] низку різновидів R-предикатів. Зокрема, в класі R-предикатів виділено 4 константних:

– тотожно істинний предикат T (маємо $T(\mathrm{T}) = {}^{V}A$ та $F(\mathrm{T}) = \varnothing$, тоді $\perp(\mathrm{T}) = \varnothing$);

– тотожно хибний предикат F (маємо $F(\mathrm{F\ T}) = {}^{V}A$ та $T(\mathrm{F}) = \varnothing$, тоді $\perp(\mathrm{F}) = \varnothing$);

– тотожно невизначений предикат $\perp$ (маємо $T(\perp) = F(\perp) = \varnothing$, тоді $\perp(\perp) = {}^{V}A$);

– тотально амбівалентний предикат Υ (маємо $T(\Upsilon) = F(\Upsilon) = {}^{V}A$, тоді $\perp(\Upsilon) = \varnothing$).

2. Пропозиційні композиції *R*-предикатів

На пропозиційному рівні предикати можна трактувати як функції вигляду $P : D \to \{T, F\}$, де D – сукупність абстрактних даних. На гранично абстрактному рівні дані трактуємо як "чорні скриньки", які неможливо відрізнити одне від одного. Тому на пропозиційному рівні предикати можна розглядати як задані на одиничному абстрактному даному. Кожний предикат на цьому даному може приймати певне істиннісне значення або бути невизначеним. Отже, на пропозиційному рівні композиції фактично працюють лише з виробленими предикатами істиннісними значеннями. Традиційно такі композиції називають логічними зв'язками.

Мінімально конкретизувавши рівень розгляду, трактуємо D як сукупність відмінних одне від одного даних, абстрактних в тому розумінні, що їх структура невідома ("чорні скриньки" різної ваги чи форми). Кожний предикат $P : D \to \{T, F\}$ тоді можна трактувати як об'єднання гранично абстрактних предикатів, заданих на одиничних даних – елементах D.

Для квазіарних предикатів множину даних D уточнюємо як ${}^{V}A$.

Основними логічними зв'язками є 1-арна композиція заперечення $\neg$ та бінарні композиції диз'юнкція $\vee$, кон'юнкція $\&$, імплікація $\to$, еквіваленція $\leftrightarrow$.

Визначення композицій $\neg$, $\vee$, $\&$, $\to$, $\leftrightarrow$ дамо, задаючи предикати $\neg(P)$, $\vee(P, Q)$, $\&(P, Q)$, $\to(P, Q)$, $\leftrightarrow(P, Q)$. Ці предикати будемо скорочено позначати $\neg P$, $P \vee Q$, $P \& Q$, $P \to Q$, $P \leftrightarrow Q$.

У випадку R-предикатів визначення пропозиційних композицій природно дати [7] через області істинності й хибності відповідних предикатів:

$$T(\neg P) = F(P);$$

$$F(\neg P) = T(P);$$

$$T(P \vee Q) = T(P) \cup T(Q);$$

$$F(P\vee Q) = F(P)\cap F(Q);$$

$$T(P\&Q) = T(P)\cap T(Q);$$

$$F(P\&Q) = F(P)\cup F(Q);$$

$$T(P\to Q) = F(P)\cup T(Q);$$

$$F(P\to Q) = T(P)\cap F(Q);$$

$$T(P\leftrightarrow Q) = (F(P)\cup T(Q))\cap(F(Q)\cup T(P));$$

$$F(P\leftrightarrow Q) = (T(P)\cap F(Q))\cup(F(P)\cap T(Q)).$$

У випадку P-предикатів визначення композицій $\neg$, $\vee$, $\&$, $\to$, $\leftrightarrow$ подібні до визначень відповідних класичних логічних зв'язок [12]. Такі пропозиційні композиції є аналогами відповідних сильних зв'язок Кліні [13], тому їх називають Клінієвими.

Для P-предикатів визначення пропозиційних композицій можна дати більш традиційно, через значення відповідних предикатів. Наведемо тут визначення композицій $\neg$, $\vee$, $\&$.

Для довільного $d\in{}^{V}A$ задаємо:

$$(\neg P)(d) = \begin{cases} T, \text{ якщо } P(d) = F, \\ F, \text{ якщо } P(d) = T, \\ \text{невизначене, якщо } P(d)\uparrow. \end{cases}$$

$$(P\vee Q)(d) = \begin{cases} T, \text{ якщо } P(d) = T \\ \quad \text{або } Q(d) = T, \\ F, \text{ якщо } P(d) = F \\ \quad \text{та } Q(d) = F, \\ \text{невизначене} \\ \quad \text{в усіх інших випадках.} \end{cases}$$

$$(P\,\&\,Q)(d) = \begin{cases} T, \text{ якщо } P(d) = T \\ \quad \text{та } Q(d) = T, \\ F, \text{ якщо } P(d) = F \\ \quad \text{або } Q(d) = F, \\ \text{невизначене} \\ \quad \text{в усіх інших випадках.} \end{cases}$$

Для P-предикатів визначення пропозиційних композицій через області істинності й хибності еквівалентні їх визначенням через значення відповідних предикатів.

Композиції $\neg$ та $\vee$ назвемо *базовими пропозиційними композиціями.*

Композиції $\to$, $\&$, $\leftrightarrow$ є *похідними*, вони виражаються через базові композиції $\neg$ та $\vee$:

$$P\&Q = \neg(\neg P\vee\neg Q);$$

$$P\to Q = \neg P\vee Q;$$

$$P\leftrightarrow Q = (P\to Q)\&(Q\to P).$$

Твердження 1. Для $\perp$-області R-предикатів маємо:

$$\perp(\neg Q) = \perp(Q);$$

$$\perp(P\vee Q) = (\perp(P)\cap\overline{T(Q)})\cup(\perp(Q)\cap\overline{T(P)}).$$

Справді, маємо

$$\perp(\neg Q) = \overline{T(\neg Q)\cup F(\neg Q)} =$$

$$= \overline{F(Q)\cup T(Q)} = \perp(Q);$$

$$\perp(P\vee Q) = \overline{T(P\vee Q)\cup F(P\vee Q)} =$$

$$= \overline{T(P)\cup T(Q)\cup(F(P)\cap F(Q)} =$$

$$= \overline{T(P)}\cap\overline{T(Q)}\cap(\overline{F(P)}\cup\overline{F(Q)}) =$$

$$= (\overline{T(P)}\cap\overline{F(P)}\cap\overline{T(Q)})\cup$$

$$\cup(\overline{T(Q)}\cap\overline{F(Q)}\cap\overline{T(P)}) =$$

$$= (\perp(P)\cap\overline{T(Q)})\cup(\perp(Q)\cap\overline{T(P)}).$$

Тут враховано, що

$$\perp(S) = \overline{T(S)}\cap\overline{F(S)}).$$

В загальному випадку R-предикатів

$$\overline{T(S)}\subseteq\perp(S)\cup F(S).$$

Водночас для P-предикатів множини $T(S)$, $F(S)$, $\perp(S)$ – диз'юнктні, звідки

$$\overline{T(S)} = \perp(S)\cup F(S).$$

Твердження 2. Для $\perp$-області P-предикатів маємо:

$$\perp(P\vee Q) = (\perp(P)\cap\perp(Q))\cup$$

$$\cup(\perp(P)\cap F(Q))\cup(\perp(Q)\cap F(P));$$

$$\perp(P\,\&\,Q) = (\perp(P)\cap\perp(Q))\cup$$

$$\cup(\perp(P)\cap T(Q))\cup(\perp(Q)\cap T(P)).$$

Справді, маємо $\perp(P\vee Q) =$

$$= (\perp(P)\cap\overline{T(Q)})\cup(\perp(Q)\cap\overline{T(P)}) =$$

$$= (\perp(P)\cap(\perp(Q)\cup F(Q)))\cup$$

$$\cup(\perp(Q)\cap(\perp(P)\cup F(P))) =$$

$$=(\perp(P)\cap\perp(Q))\cup(\perp(P)\cap F(Q))\cup.$$

$$\cup(\perp(Q)\cap F(P)).$$

Це означає:

$$(P\vee Q)(d)\uparrow \Leftrightarrow P(d)\uparrow \text{ та } Q(d)\uparrow \text{ або}$$

$$P(d)\uparrow \text{ та } Q(d)=F \text{ або } P(d)=F \text{ та } Q(d)\uparrow.$$

Подібним чином отримуємо вираз для $\perp(P\&Q)$.

Зазначимо, що для найзагальнішого класу недетермінованих предикатів – *GND*-предикатів [14] – маємо такі ж подання $\perp(P\vee Q)$ та $\perp(P\&Q)$, як і для *P*-предикатів. Водночас, на відміну від *R*-предикатів, для *GND*-предикатів можливо $\perp(P)\cap T(P)\neq\varnothing$ та $\perp(P)\cap F(P)\neq\varnothing$.

Основні властивості пропозиційних композицій *R*-предикатів аналогічні властивостям відповідних класичних логічних зв'язок [6, 12].

Клас *P*-предикатів замкнений [7, 9] щодо композицій $\neg$, $\vee$, &, $\rightarrow$, $\leftrightarrow$.

3. Композиція предикатного доповнення

Характерною особливістю LC є наявність спеціальної 1-арної композиції предикатного доповнення $\tilde{}$. Ця композиція задається так:

$$(\tilde{}P)(d)=\begin{cases} T, & \text{якщо } P(d)\uparrow, \\ \text{невизначене}, & \text{якщо } P(d)\downarrow. \end{cases}$$

Отже, композиція $\tilde{}$ немонотонна.

Композицію $\tilde{}$ можна задати через *T*-область і *F*-область відповідного предиката:

$$T(\tilde{}P)=\perp(P)=\overline{T(P)\cup F(P)}=\overline{T(P)}\cap\overline{F(P)},$$

$$F(\tilde{}P)=\varnothing.$$

Тоді маємо $\perp(\tilde{}P)=T(P)\cup F(P)$.

Класи *P*-предикатів та *R*-предикатів замкнені щодо композиції $\tilde{}$.

Для довільного *R*-предиката *Q* маємо, що $\tilde{}Q$ є *P*-предикатом, тому

клас T-предикатів незамкнений щодо $\tilde{}$.

Отже, для *T*-предикатів LC не мають смислу.

Таким чином, можна розглядати загальний клас LC – логіки *R*-предикатів з композицією предикатного доповнення, та їх підклас LPC – логіки *P*-предикатів з композицією предикатного доповнення.

Для LC пропозиційного рівня маємо загальний клас PLC *R*-предикатів та його підклас PLPC – PLC *P*-предикатів.

Композиції $\neg$, $\vee$, $\tilde{}$ – це *базові* композиції PLC.

4. Пропозиційні композиційні алгебри та мови LC

Семантичною основою LC є композиційні предикатні системи вигляду $({}^{V}A, Pr^{V-A}, C_{LC})$, де C_{LC} – множина базових композицій. Така композиційна система задає дві алгебри: алгебру (алгебраїчну систему) даних $({}^{V}A, Pr^{V-A})$ та композиційну алгебру предикатів $({}^{V}A, Pr^{V-A}, C_{LC})$.

Композиційну систему вигляду $({}^{V}A, Pr^{V-A}, C_{PLC})$, де множина композицій $C_{PLC}=\{\neg,\vee,\tilde{}\}$, назвемо *пропозиційною композиційною LC-системою R-предикатів.*

Алгебру $APLC=(Pr^{V-A}, C_{PLC})$ назвемо *пропозиційною композиційною LC-алгеброю R-предикатів*.

Клас *P*-предикатів замкнений щодо композиції $\tilde{}$. Тому можна говорити про те, що в алгебрі *APLC* виділена підалгебра $APLCP=(PrP^{V-A}, C_{PLC})$ – *пропозиційна композиційна LC-алгебра P-предикатів*.

Розглянемо основні властивості композицій PLC.

Твердження 3. Із визначень отримуємо такі властивості композиції $\tilde{}$:

$$\tilde{}\neg P=\tilde{}P;$$

$$\tilde{}\tilde{}\tilde{}P=\tilde{}P;$$

$$\tilde{}\tilde{}\tilde{}\tilde{}P=\tilde{}\tilde{}P.$$

Для *P*-предикатів додатково маємо:

$$\tilde{}\tilde{}P=P\vee\neg P.$$

Таким чином, багатократне застосування композиції $\tilde{}$ зводиться до 1-кратного та до 2-кратного її застосування.

Із визначень отримуємо:

$$(\neg\tilde{}P)(d) = \begin{cases} F, & \text{якщо } P(d)\uparrow, \\ \text{невизначене}, & \text{якщо } P(d)\downarrow \end{cases}$$

Маємо $F(\neg\tilde{}P) = T(\tilde{}P) = \bot(P)$, звідки отримуємо:

$$F(\neg\tilde{}P) = \overline{T(P) \cup F(P)} = \overline{T(P)} \cap \overline{F(P)},$$

$$T(\neg\tilde{}P) = \varnothing.$$

Властивості пропозиційних композицій в LC аналогічні властивостям пропозиційних композицій R-предикатів у традиційних логіках квазіарних предикатів [7].

Твердження 4. Маємо такі основні властивості композицій $\neg$, $\vee$, $\&$ в LC.

1) комутативність $\vee$ та $\&$:

$$P \vee Q = Q \vee P;$$

$$P \& Q = Q \& P.$$

2) асоціативність $\vee$ та $\&$:

$$(P \vee Q) \vee R = P \vee (Q \vee R);$$

$$(P \& Q) \& R = P \& (Q \& R).$$

3) дистрибутивність $\vee$ відносно $\&$ та $\&$ відносно $\vee$:

$$(P \vee Q) \& R = (P \& R) \vee (Q \& R);$$

$$(P \& Q) \vee R = (P \vee R) \& (Q \vee R).$$

4) ідемпотентність $\vee$ та $\&$:

$$P = P \vee P;$$

$$P = P \& P.$$

5) зняття подвійного заперечення:

$$\neg\neg P = P.$$

6) закон контрапозиції:

$$P \to Q = \neg Q \to \neg P.$$

7) закони де Моргана:

$$\neg(P \vee Q) = (\neg P) \& (\neg Q);$$

$$\neg(P \& Q) = (\neg P) \vee (\neg Q).$$

8) закони поглинання:

$$P = P \vee (P \& Q);$$

$$P = P \& (P \vee Q).$$

Опишемо мови PLC.

Алфавіт мови:

– множина V предметних імен, або змінних,

– множина Ps предикатних символів,

– множина $Cs_{PLC} = \{\neg, \vee, \tilde{}\}$ символів базових композицій.

Множину Fr формул мови визначаємо так.

Маємо $Ps \subseteq Fr$, а далі задаємо індуктивно:

$$\Phi, \Psi \in Fr \Rightarrow \neg\Phi, \vee\Phi\Psi, \tilde{}\Phi \in Fr.$$

Інтерпретуємо мову на композиційних системах $({}^{V}A, Pr^{V-A}, C_{PLC})$.

Задаємо тотальне однозначне відображення інтерпретації предикатних символів $I : Ps \to Pr^{V-A}$, яке далі продовжуємо до відображення інтерпретації формул $I : Fr \to Pr^{V-A}$:

$$I(\neg\Phi) = \neg(I(\Phi)),$$

$$I(\vee\Phi\Psi) = \vee(I(\Phi), I(\Psi)),$$

$$I(\tilde{}\Phi) = \tilde{}(I(\Phi)).$$

Нехай $\boldsymbol{J} = (A, I)$ – інтерпретація. Предикат $\boldsymbol{J}(\Phi)$ позначаємо Φ_J.

Композиції пропозиційного рівня беруть до уваги лише вироблені предикатами істиннісні значення, тому кожне відображення інтерпретації $I : Ps \to Pr^{V-A}$ індукує сукупність істиннісних оцінок (взагалі кажучи, часткових) таким чином.

Для кожного $d \in {}^{V}A$ задаємо істиннісну оцінку $\upsilon_d : Ps \to \{T, F\}$ так:

$$\upsilon_d(p) = p_J(d).$$

Тоді для кожної $\Phi \in Fr$ маємо

$$\upsilon_d(\Phi) = \Phi_J(d).$$

Зокрема, якщо $\Phi_J(d)\uparrow$, то й $\upsilon_d(\Phi)\uparrow$.

Виділення класів квазіарних предикатів виділяє відповідні класи інтерпретацій. Такі класи інтерпретацій називають [9] семантиками.

Клас T-предикатів незамкнений щодо $\tilde{}$, тому для LC змістовними є R-се-

мантика та *P*-семантика, будемо їх відповідно позначати $\boldsymbol{R_C}$ та $\boldsymbol{P_C}$.

Фіксуючи пропозиційний рівень, для PLC отримуємо семантики $\boldsymbol{R_{PC}}$ та $\boldsymbol{P_{PC}}$.

Зрозуміло, що наведені вище властивості логічних зв'язок та композиції предикатного доповнення можна подати із використанням формул мови PLC.

5. Відношення логічного наслідку за умов невизначеності

На множинах формул LC можна визначити низку відношень логічного наслідку.

В даній роботі введемо та дослідимо відношення *неспростовнісного логічного наслідку за умов невизначеності*. Таке відношення є узагальненням відомого [7, 9] відношення неспростовнісного логічного наслідку для традиційних логік квазіарних предикатів.

Нехай деяка $\Sigma \subseteq Fr$; нехай $\boldsymbol{J}$ – інтерпретація. Далі позначаємо:

$$\bigcap_{\theta\in\Sigma} T(\theta_J) \text{ як } T^{\frown}(\Sigma_J),$$

$$\bigcap_{\theta\in\Sigma} F(\theta_J) \text{ як } F^{\frown}(\Sigma_J),$$

$$\bigcap_{\theta\in\Sigma} \perp(\theta_J) \text{ як } \perp^{\frown}(\Sigma_J).$$

Множина Σ може бути порожньою. У випадку, коли $\Sigma = \varnothing$, маємо:

$$T^{\frown}(\Sigma) = T^{\frown}(\varnothing) = F^{\frown}(\Sigma) = F^{\frown}(\varnothing) = \\ = \perp^{\frown}(\Sigma) = \perp^{\frown}(\varnothing) = {}^{V}A.$$

Спочатку вводимо відношення *неспростовнісного наслідку за умов невизначеності при певній інтерпретації*.

Нехай $\vartheta, \Phi, \Psi \in Fr$. Неформально те, що Ψ є неспростовнісним наслідком Φ за умови невизначеності ϑ при інтерпретації $\boldsymbol{J}$, має означати: "якщо ϑ_J невизначене, то неможливо, що Φ_J істинне та Ψ_J хибне". Це можна уточнити так: "для кожного $d \in {}^{V}A$ якщо $\vartheta_J(d)\uparrow$, то неможливо $\Phi_J(d) = T$ та $\Psi_J(d) = F$".

Останнє твердження рівносильне такому: "для кожного $d \in {}^{V}A$ неправильно $\vartheta_J(d)\uparrow$ або неправильно $\Phi_J(d) = T$ та $\Psi_J(d) = F$". Це в свою чергу рівносильне такому: "для кожного $d \in {}^{V}A$ неправильно, що $\vartheta_J(d)\uparrow$, та $\Phi_J(d) = T$ та $\Psi_J(d) = F$".

Таким чином, початкове твердження звелося до рівносильної умови: "не існує $d \in {}^{V}A$ таких, що $d \in \perp(\vartheta_J) \cap T(\Phi_J) \cap F(\Psi_J)$", що дає умову $\perp(\vartheta_J) \cap T(\Phi_J) \cap F(\Psi_J) = \varnothing$.

Подібні міркування можна провести для множин формул. Нехай $\Gamma, \mathrm{U}, \Delta \subseteq Fr$. Неформально те, що Δ є неспростовнісним наслідком Γ за умов невизначеності U при інтерпретації $\boldsymbol{J}$, має означати: "для кожного $d \in {}^{V}A$, якщо $\varphi_J(d)\uparrow$ для всіх $\varphi_J \in \mathrm{U}$, то неможливо $\xi_J(d) = T$ для всіх $\xi \in \Gamma$ та $\psi_J(d) = F$ для всіх $\psi \in \Delta$".

Це твердження рівносильне такому: "для кожного $d \in {}^{V}A$, якщо $d \in \perp^{\frown}(\mathrm{U}_J)$, то неможливо $d \in T^{\frown}(\Gamma_J)$ та $d \in F^{\frown}(\Delta_J)$".

Останнє твердження рівносильне такому: "для кожного $d \in {}^{V}A$ неправильно, що $d \in \perp^{\frown}(\mathrm{U}_J)$ та $d \in T^{\frown}(\Gamma_J) \cap F^{\frown}(\Delta_J)$, тобто неправильно, що $d \in \perp^{\frown}(\mathrm{U}_J) \cap T^{\frown}(\Gamma_J) \cap \\ \cap F^{\frown}(\Delta_J)$".

Отже, те, що Δ є неспростовнісним наслідком Γ за умов невизначеності U, звелося до такої умови: "не існує $d \in {}^{V}A$ таких, що $d \in \perp^{\frown}(\mathrm{U}_J) \cap T^{\frown}(\Gamma_J) \cap F^{\frown}(\Delta_J)$".

Це означає:

$$T^{\frown}(\Gamma_J) \cap \perp^{\frown}(\mathrm{U}_J) \cap F^{\frown}(\Delta_J) = \varnothing.$$

Той факт, що Δ є неспростовнісним наслідком Γ за умов невизначеності U при інтерпретації $\boldsymbol{J}$, будемо позначати $\mathrm{U} / \Gamma_J \models_{IR}^{\perp} \Delta$.

Таким чином, ми прийшли до наступного визначення:

$$\mathrm{U} / \Gamma_J \models_{IR}^{\perp} \Delta, \text{ якщо}$$

$$T^{\frown}(\Gamma_J) \cap \perp^{\frown}(\mathrm{U}_J) \cap F^{\frown}(\Delta_J) = \varnothing.$$

Зокрема, якщо $\mathrm{U} = \varnothing$, то отримуємо умову $T^{\frown}(\Gamma_J) \cap F^{\frown}(\Delta_J) = \varnothing$ – умову відношення неспростовнісного наслідку $\Gamma_J \models_{IR} \Delta$.

Відношення неспростовнісного логічного наслідку в LC за умов невизначеності визначаємо традиційно.

Δ є неспростовнісним логічним наслідком Γ за умов невизначеності U, що позначимо $\mathrm{U} / \Gamma \models_{IR}^{\perp} \Delta$, якщо $\mathrm{U} / \Gamma_J \models_{IR}^{\perp} \Delta$ для кожної інтерпретації $\boldsymbol{J} \in \boldsymbol{P_C}$.

Зокрема, якщо $U = \varnothing$, то отримуємо відоме відношення $\Gamma \models_{IR} \Delta$.

6. Властивості відношення $\models_{IR}^{\perp}$

Опишемо основні властивості відношення $\models_{IR}^{\perp}$ на пропозиційному рівні.

Із визначень отримуємо властивість монотонності M:

M) нехай $\Gamma \subseteq \Lambda$, $U \subseteq W$, та $\Delta \subseteq \Sigma$; тоді $U / \Gamma \models_{IR}^{\perp} \Delta \Rightarrow W / \Lambda \models_{IR}^{\perp} \Sigma$.

Теорема 1. Маємо такі властивості гарантованої наявності відношення логічного наслідку $\models_{IR}^{\perp}$:

C) $U / \Phi, \Gamma \models_{IR}^{\perp} \Delta, \Phi$;

C_{UL}) $U, \Phi / \Phi, \Gamma \models_{IR}^{\perp} \Delta$;

C_{UR}) $U, \Phi / \Gamma \models_{IR}^{\perp} \Delta, \Phi$;

$C_{\sim}$) $U / \Gamma \models_{IR}^{\perp} \Delta, {\sim}\Phi$.

Доведемо властивість C.

Для P-предикатів $T(\Phi_J) \cap F(\Phi_J) = \varnothing$. Тоді для кожної інтерпретації $\boldsymbol{J}$ маємо $T(\Phi_J) \cap T^{\frown}(\Gamma_J) \cap \perp^{\frown}(U_J) \cap F^{\frown}(\Delta_J) \cap F(\Phi_J) = \varnothing$, звідки $U / \Phi, \Gamma \models_{IR}^{\perp} \Delta, \Phi$.

Доведемо властивість C_{UL}.

Маємо $\perp(\Phi_J) \cap T(\Phi_J) = \varnothing$. Звідси для кожної інтерпретації $\boldsymbol{J}$ отримуємо $T(\Phi_J) \cap T^{\frown}(\Gamma_J) \cap \perp^{\frown}(U_J) \cap \perp(\Phi_J) \cap F^{\frown}(\Delta_J) = \varnothing$, звідки $U, \Phi / \Phi, \Gamma \models_{IR}^{\perp} \Delta$.

Доведемо властивість C_{UR}.

Маємо $\perp(\Phi_J) \cap F(\Phi_J) = \varnothing$. Звідси для кожної інтерпретації $\boldsymbol{J}$ отримуємо $T^{\frown}(\Gamma_J) \cap \perp^{\frown}(U_J) \cap \perp(\Phi_J) \cap F^{\frown}(\Delta_J) \cap F(\Phi_J) = \varnothing$, звідки $U, \Phi / \Gamma \models_{IR}^{\perp} \Delta, \Phi$.

Доведемо властивість $C_{\sim}$. Для кожної інтерпретації $\boldsymbol{J}$ маємо $F({\sim}\Phi_J) = \varnothing$, тому $T^{\frown}(\Gamma_J) \cap \perp^{\frown}(U_J) \cap F^{\frown}(\Delta_J) \cap F({\sim}\Phi_J) = \varnothing$, звідки $U / \Gamma \models_{IR}^{\perp} \Delta, {\sim}\Phi$.

Теорема 2. Для відношення $\models_{IR}^{\perp}$ маємо наступні властивості декомпозиції формул:

$\neg_L$) $U / \neg\Phi, \Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow U / \Gamma \models_{IR}^{\perp} \Delta, \Phi$.

$\neg_R$) $U / \Gamma \models_{IR}^{\perp} \Delta, \neg\Phi \Leftrightarrow U / \Phi, \Gamma \models_{IR}^{\perp} \Delta$.

$\vee_L$) $U / \Phi \vee \Psi, \Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow$

$\Leftrightarrow U / \Phi, \Gamma \models_{IR}^{\perp} \Delta$ та $U / \Psi, \Gamma \models_{IR}^{\perp} \Delta$.

$\vee_R$) $U / \Gamma \models_{IR}^{\perp} \Delta, \Phi \vee \Psi \Leftrightarrow$

$\Leftrightarrow U / \Gamma \models_{IR}^{\perp} \Delta, \Phi, \Psi$.

$\neg_U$) $U, \neg\vartheta / \Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow U, \vartheta / \Gamma \models_{IR}^{\perp} \Delta$.

$\vee_U$) $U, \Phi \vee \vartheta / \Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow$

$\Leftrightarrow U, \Phi, \vartheta / \Gamma \models_{IR}^{\perp} \Delta$ та

$U, \Phi / \Gamma \models_{IR}^{\perp} \vartheta, \Delta$ та $U, \vartheta / \Gamma \models_{IR}^{\perp} \Phi, \Delta$.

${\sim}_U$) $U, {\sim}\Phi / \Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow$

$\Leftrightarrow U / \Phi, \Gamma \models_{IR}^{\perp} \Delta$ та $U / \Gamma \models_{IR}^{\perp} \Delta, \Phi$.

${\sim}_L$) $U / {\sim}\Phi, \Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow$

$\Leftrightarrow U, \Phi / \Gamma \models_{IR}^{\perp} \Delta$.

Властивості $\neg_L$, $\neg_R$, $\vee_L$, $\vee_R$ аналогічні відповідним властивостям відношення $\models_{IR}$ [7, 9] та доводяться подібним чином. Новими для LC є специфічні властивості $\neg_U$, $\vee_U$, ${\sim}_L$, ${\sim}_U$.

Доведемо властивість $\neg_U$.

Для кожної інтерпретації $\boldsymbol{J}$ маємо

$\perp(\neg\vartheta_J) = \perp(\vartheta_J)$, звідки $U, \neg\vartheta / \Gamma_J \models_{IR}^{\perp} \Delta \Leftrightarrow$

$\Leftrightarrow T^{\frown}(\Gamma_J) \cap \perp^{\frown}(U_J) \cap \perp(\neg\vartheta_J) \cap F^{\frown}(\Delta_J) = \varnothing$

$\Leftrightarrow T^{\frown}(\Gamma_J) \cap \perp^{\frown}(U_J) \cap \perp(\vartheta_J) \cap F^{\frown}(\Delta_J) = \varnothing \Leftrightarrow$

$U, \vartheta / \Gamma_J \models_{IR}^{\perp} \Delta$.

Властивість $\vee_U$ випливає з того, що для P-предикатів маємо:

$$\perp(\Phi_J \vee \vartheta_J) = (\perp(\Phi_J) \cap \perp(\vartheta_J)) \cup$$

$$\cup(\perp(\Phi_J) \cap F(\vartheta_J)) \cup (F(\Phi_J) \cap \perp(\vartheta_J)).$$

При цьому враховуємо таке теоретико-множинне співвідношення:

$$L \cap ((\Phi_{\perp} \cap \vartheta_{\perp}) \cup (\Phi_{\perp} \cap \vartheta_F) \cup (\Phi_F \cap \vartheta_{\perp})) =$$

$$= (L \cap \Phi_{\perp} \cap \vartheta_{\perp}) \cup (L \cap \Phi_{\perp} \cap \vartheta_F) \cup$$

$$\cup (L \cap \Phi_F \cap \vartheta_{\perp}) = \varnothing \Leftrightarrow$$

$\Leftrightarrow L \cap \Phi_{\perp} \cap \vartheta_{\perp} = \varnothing$

та $L \cap \Phi_{\perp} \cap \vartheta_F = \varnothing$

та $L \cap \Phi_F \cap \vartheta_{\perp} = \varnothing$.

Таким чином, для кожної інтерпретації $\boldsymbol{J}$ маємо:

$U, \Phi \vee \vartheta / \Gamma_J \models_{IR}^{\perp} \Delta \Leftrightarrow$

$\Leftrightarrow T^{\frown}(\Gamma_J) \cap \perp^{\frown}(U_J) \cap \perp(\Phi \vee \vartheta_J) \cap F^{\frown}(\Delta_J) = \varnothing$

$$\Leftrightarrow T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap((\perp(\Phi_J)\cap\perp(\vartheta_J))\cup$$

$$\cup(\perp(\Phi_J)\cap F(\vartheta_J))\cup(F(\Phi_J)\cap\perp(\vartheta_J)))\cap F^{\frown}(\Delta_J)=\varnothing$$

$$\Leftrightarrow T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap\perp(\Phi_J)\cap\perp(\vartheta_J)\cap F^{\frown}(\Delta_J)=\varnothing,$$

$$\text{та } T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap\perp(\Phi_J)\cap F(\vartheta_J)\cap F^{\frown}(\Delta_J)=\varnothing,$$

$$\text{та } T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap F(\Phi_J)\cap\perp(\vartheta_J)\cap F^{\frown}(\Delta_J)=\varnothing$$

$$\Leftrightarrow U,\Phi,\vartheta/\Gamma_J|=_{IR}^{\perp}\Delta \text{ та } U,\Phi/\Gamma_J|=_{IR}^{\perp}\vartheta,\Delta$$

$$\text{та } U,\vartheta/\Gamma_J|=_{IR}^{\perp}\Phi,\Delta.$$

Властивість $\sim_U$ випливає із такого:

$$\perp(\sim\Phi_J)=T(\Phi_J)\cup F(\Phi_J).$$

При цьому враховуємо наступне теоретико-множинне співвідношення:

$$L\cap(L_T\cup L_F)=(L\cap L_T)\cup(L\cap L_F)=\varnothing \Leftrightarrow$$

$$\Leftrightarrow L\cap L_T=\varnothing \text{ та } L\cap L_F=\varnothing.$$

Таким чином, для кожної інтерпретації ***J*** маємо:

$$U,\sim\Phi/\Gamma_J|=_{IR}^{\perp}\Delta \Leftrightarrow$$

$$\Leftrightarrow T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap\perp(\sim\Phi_J)\cap F^{\frown}(\Delta_J)=\varnothing$$

$$\Leftrightarrow T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap(T(\Phi_J)\cup F(\Phi_J))\cap$$

$$\cap F^{\frown}(\Delta_J)=\varnothing \Leftrightarrow$$

$$\Leftrightarrow T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap T(\Phi_J)\cap F^{\frown}(\Delta_J)=\varnothing \text{ та}$$

$$T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap F(\Phi_J)\cap F^{\frown}(\Delta_J)=\varnothing \Leftrightarrow$$

$$\Leftrightarrow U/\Phi,\Gamma_J|=_{IR}^{\perp}\Delta \text{ та } U/\Gamma_J|=_{IR}^{\perp}\Delta,\Phi.$$

Властивість $\sim_L$ випливає із такого:

$$T(\sim\Phi_J)=\perp(\Phi_J).$$

Тоді для кожної інтерпретації ***J*** маємо:

$$U/\sim\Phi,\Gamma|=_{IR}^{\perp}\Delta \Leftrightarrow$$

$$\Leftrightarrow T(\sim\Phi_J)\cap T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap F^{\frown}(\Delta_J)=\varnothing$$

$$\Leftrightarrow T^{\frown}(\Gamma_J)\cap\perp^{\frown}(U_J)\cap\perp(\Phi_J)\cap F^{\frown}(\Delta_J)=\varnothing$$

$$\Leftrightarrow U,\Phi/\Gamma|=_{IR}^{\perp}\Delta.$$

7. Пропозиційне секвенційне числення для відношення $|=_{IR}^{\perp}$

Для відношення $|=_{IR}^{\perp}$ в PLC побудуємо числення секвенційного типу.

Таке числення назвемо PP_C.

Секвенції будемо трактувати як множини формул, специфікованих (відмічених) символами $\vdash, \dashv, \perp$.

Формули із Γ (вони відмічені символом $\vdash$) назвемо *T*-формулами, формули із Δ (вони відмічені символом $\dashv$) назвемо *F*-формулами, а формули із U (вони відмічені символом $\perp$) – ⊥-формулами, Це відповідає семантичному трактуванню формул із Γ як істинних, формул із Δ – як хибних. а формул із U – як невизначених.

Позначаємо секвенції як $\vdash\Gamma_{\perp}U\dashv\Delta$, скорочено як Σ.

Виведення в секвенційних численнях має вигляд дерева, вершинами якого є секвенції. Такі дерева називають секвенційними.

Правилами виведення секвенційних числень є секвенційні форми. Вони є синтаксичними аналогами семантичних властивостей відповідних відношень логічного наслідку.

Секвенційні форми будемо записувати у вигляді $\frac{\Sigma}{\Omega}$ або $\frac{\Sigma\ \ \Lambda}{\Omega}$, або $\frac{\Sigma\ \ \Lambda\ \ \Theta}{\Omega}$.

Секвенції над рискою називають засновками, під рискою – висновками. У нашому випадку засновки – це секвенції, зіставлені правим частинам відповідних семантичних властивостей, висновки – це секвенції, зіставлені їх лівим частинам.

Дамо індуктивне визначення секвенційного дерева.

1. Секвенція Σ утворює *тривіальне* секвенційне дерево з єдиною вершиною Σ, яка є коренем дерева.

2. Нехай α, β, γ – секвенційні дерева, коренями яких є відповідно Σ, Υ, Θ; нехай $\frac{\Sigma}{\Omega}, \frac{\Sigma\ \ \Upsilon}{\Omega}, \frac{\Sigma\ \ \Lambda\ \ \Theta}{\Omega}$ – секвенційні форми. Тоді $\frac{\alpha}{\Omega}\ |\ , \frac{\alpha\ \ \beta}{\Omega}\ \backslash/\ , \frac{\alpha\ \ \beta\ \ \gamma}{\Omega}\ \backslash\,|\,/$ – секвенційні дерева з коренем Ω.

Аксіомами секвенційного числення є замкнені секвенції.

Поняття замкненої секвенції уточнюється так, що має виконуватись умова:

якщо секвенція $\vdash\Gamma_{\perp}U\dashv\Delta$ замкнена,

то $U/\Gamma \models_{IR}^{\perp} \Delta$.

Секвенція Σ *вивідна*, або *має виведення*, якщо існує *замкнене секвенційне дерево* з коренем Σ. Таке дерево називають виведенням секвенції Σ.

Тривіальне секвенційне дерево замкнене, якщо це замкнена секвенція.

Нетривіальне секвенційне дерево замкнене, якщо кожний його лист (кінцева вершина, відмінна від кореня) – замкнена секвенція.

Секвенційне числення визначається *базовими секвенційними формами* та *умовами замкненості секвенції*.

Наведемо умову замкненості секвенції у численні PP_C.

Секвенція $\vdash\Gamma_{\perp}U\dashv\Delta$ *замкнена*, якщо виконується умова $C \vee C_{UL} \vee C_{UR} \vee C_{\sim}$.

Тут:

C) існує Φ: $\Phi\in\Gamma$ та $\Phi\in\Delta$;

C_{UL}) існує Φ: $\Phi\in\Gamma$ та $\Phi\in U$;

C_{UR}) існує Φ: $\Phi\in\Delta$ та $\Phi\in U$.

$C_{\sim}$) існує Φ: $\sim\Phi\in\Delta$.

Замкненість секвенції $\vdash\Gamma_{\perp}U\dashv\Delta$ гарантує наявність логічного наслідку $U/\Gamma \models_{IR}^{\perp} \Delta$.

Це безпосередньо випливає з властивостей гарантованої наявності логічного наслідку C, C_{UL}, C_{UR}, $C_{\sim}$.

Базові секвенційні форми PP_C-числень – це форми декомпозиції, вони індукуються відповідними властивостями декомпозиції формул:

$$\vdash\neg\ \frac{\dashv\Phi,\ \Sigma}{\vdash\neg\Phi,\ \Sigma};\qquad \dashv\neg\ \frac{\vdash\Phi,\ \Sigma}{\dashv\neg\Phi,\ \Sigma};$$

$$\vdash\vee\ \frac{\vdash\Phi,\ \Sigma\quad\vdash\Psi,\ \Sigma}{\vdash\Phi\vee\Psi,\ \Sigma};\qquad \dashv\vee\ \frac{\dashv\Phi,\ \dashv\Psi,\ \Sigma}{\dashv\Phi\vee\Psi,\ \Sigma};$$

$$\perp\neg\ \frac{\perp\Phi,\ \Sigma}{\perp\neg\Phi,\ \Sigma};$$

$$\perp\vee\ \frac{\perp\Phi,\perp\vartheta,\ \Sigma\quad\perp\Phi,\dashv\vartheta,\ \Sigma\quad\dashv\Phi,\perp\vartheta,\ \Sigma}{\perp\Phi\vee\vartheta,\ \Sigma};$$

$$\perp\sim\ \frac{\vdash\Phi,\ \Sigma\quad\dashv\Phi,\ \Sigma}{\perp\sim\Phi,\ \Sigma};\qquad \vdash\sim\ \frac{\perp\Phi,\ \Sigma}{\vdash\sim\Phi,\ \Sigma}.$$

Форми декомпозиції індукуються відповідними властивостями декомпозиції формул $\neg_L$, $\neg_R$, $\vee_L$, $\vee_R$, $\neg_U$, $\vee_U$, $\sim_U$, $\sim_L$.

На базі властивостей відношення $\models_{IR}^{\perp}$ маємо основну властивість секвенційних форм PP_C-числень.

Теорема 3.

1. Нехай $\dfrac{\vdash\Lambda_{\perp}W\dashv K}{\vdash\Gamma_{\perp}U\dashv\Delta}$ – базова секвенційна форма.

Тоді:

a) $U/\Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow W/\Lambda \models_{IR}^{\perp} K$;

b) $U/\Gamma \not\models_{IR}^{\perp} \Delta \Leftrightarrow W/\Lambda \not\models_{IR}^{\perp} K$;

2. Нехай $\dfrac{\vdash\Lambda_{\perp}W\dashv K\quad\vdash X_{\perp}V\dashv Z}{\vdash\Gamma_{\perp}U\dashv\Delta}$ – базова секвенційна форма. Тоді:

a) $U/\Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow W/\Lambda \models_{IR}^{\perp} K$ та $V/X \models_{IR}^{\perp} Z$;

b) $U/\Gamma \not\models_{IR}^{\perp} \Delta \Leftrightarrow W/\Lambda \not\models_{IR}^{\perp} K$ або $V/X \not\models_{IR}^{\perp} Z$.

3. Нехай $\dfrac{\vdash\Lambda_{\perp}W\dashv K\quad\vdash X_{\perp}V\dashv Z\quad\vdash M_{\perp}Y\dashv N}{\vdash\Gamma_{\perp}U\dashv\Delta}$ – базова секвенційна форма.

Тоді:

a) $U/\Gamma \models_{IR}^{\perp} \Delta \Leftrightarrow W/\Lambda \models_{IR}^{\perp} K$ та $V/X \models_{IR}^{\perp} Z$ та $Y/M \models_{IR}^{\perp} N$;

b) $U/\Gamma \not\models_{IR}^{\perp} \Delta \Leftrightarrow W/\Lambda \not\models_{IR}^{\perp} K$ або $V/X \not\models_{IR}^{\perp} Z$ або $Y/M \not\models_{IR}^{\perp} N$.

Для побудованого секвенційного числення справджуються теореми коректності та повноти. Доведення цих теорем будуть наведені в наступних статтях.

Висновки

В роботі досліджено нові програмно-орієнтовані логічні формалізми – логіки часткових квазіарних предикатів з предикатним доповненням. Такі логіки названі LC. Характерною особливістю цих логік є наявність спеціальної немонотонної опера-

ції (композиції) предикатного доповнення. Подібні операції використовуються в різних варіантах логік Флойда – Хоара з частковими перед- та після-умовами.

Основна увага в роботі зосереджена на вивченні LC пропозиційного рівня – PLC. Досліджено властивості композиції предикатного доповнення. Описано пропозиційні композиційні алгебри та мови PLC.

Для LC однозначних предикатів запропоновано та досліджено відношення неспростовнісного логічного наслідку за умов невизначеності. На основі властивостей цього відношення для PLC однозначних предикатів побудовано числення секвенційного типу.

Література

1. Handbook of Logic in Computer Science. Edited by S. Abramsky, Dov M. Gabbay and T. S. E. Maibaum. Oxford University Press, Vol. 1–5. 1993–2000.
2. Avron A., Zamansky A. Non-deterministic semantics for logical systems, in Handbook of Philosophical Logic, D.M. Gabbay, F. Guenthner (eds.), 2 nd ed. Vol. 16. (2011). Springer Netherlands. P. 227–304.
3. Gries D., Schneider F.B. Avoiding the undefined by underspecification. Springer Berlin Heidelberg. 1995. P. 366–373.
4. Hähnle R. Many-valued logic, partiality, and abstraction in formal specification languages. Logic Journal of the IGPL. **13** (2005). P. 415–433.
5. Jones C. Reasoning about partial functions in the formal development of programs. In: Proceedings of AVoCS'05. Vol. 145. Elsevier, Electronic Notes in Theoretical Computer Science (2006). P. 3–25.
6. Нікітченко М.С. Математична логіка та теорія алгоритмів. К.: ВПЦ Київський університет. 2008. 528 с.
7. Нікітченко М.С. , Шкільняк С.С. Прикладна логіка. К.: ВПЦ Київський університет. 2013. 278 с.
8. Nikitchenko M., Shkilniak S. Semantic Properties of Logics of Quasiary Predicates. Workshop on Foundations of Informatics: Proceedings FOI-2015. Chisinau, Moldova. P. 180–197.
9. Нікітченко М.С., Шкільняк О.С., Шкільняк С.С. Чисті першопорядкові логіки квазіарних предикатів. *Проблеми програмування*. 2016. № 2–3. С. 73–86.
10. Mykola S. Nikitchenko and Stepan S. Shkilniak. Algebras and logics of partial quasiary predicates. Algebra and Discrete Mathematics, Volume 23 (2017). Number 2, P. 263–278.
11. Ivanov I., Nikitchenko M. On the sequence rule for the Floyd-Hoare logic with partial pre- and post-conditions. In Proceedings of the 14th International Conference on ICT. 2018. Vol. 2104 of CEUR Workshop Proc. P. 716–724.
12. Клини С. Математическая логика. М.: Мир, 1973. 480 с.
13. Kleene S.C. Introductions to Metamathematics. Van Nostrand, Princeton, 1952.
14. Нікітченко М.С., Шкільняк О.С., Шкільняк С.С. Логіки загальних недетермінованих предикатів: семантичні аспекти. *Проблеми програмування*. 2018. № 2–3. С. 31–45.

References

1. Abramsky S., Gabbay D. and Maibaum T. (editors). (1993–2000). Handbook of Logic in Computer Science Oxford University Press, Vol. 1–5.
2. Avron A. and Zamansky A. (2011). Non-deterministic semantics for logical systems. In Handbook of Philosophical Logic, D.M. Gabbay, F. Guenthner (eds.), 2nd ed., Vol. 16, Springer Netherlands. P. 227–304.
3. Gries D. and Schneider F. (1995). Avoiding the undefined by underspecification. Springer Berlin Heidelberg.
4. Hähnle R. (2005). Many-valued logic, partiality, and abstraction in formal specification languages. In Logic Journal of the IGPL, **13**. P. 415–433.
5. Jones C. (2006). Reasoning about partial functions in the formal development of programs. In Proceedings of AVoCS'05. Vol. 145. Elsevier, Electronic Notes in Theoretical Computer Science. P. 3–25.
6. Nikitchenko M. and Shkilniak S. (2008). Mathematical logic and theory of algorithms. Kyiv: VPC Kyivskyi Universytet (in ukr).
7. Nikitchenko M. and Shkilniak S. (2013). Applied logic. Kyiv: VPC Kyivskyi Universytet (in ukr).
8. Nikitchenko M. and Shkilniak S. (2015). Semantic Properties of Logics of Quasiary Predicates. In Workshop on Foundations of

Informatics: Proceedings FOI-2015. Chisinau, Moldova. P. 180–197.
9. Nikitchenko M., Shkilniak O. and Shkilniak S. (2016). Pure first-order logics of quasiary predicates. In Problems in Progamming. N 2–3. P. 73–86 (in ukr).
10. Nikitchenko M. and Shkilniak S. (2017). Algebras and logics of partial quasiary predicates. In Algebra and Discrete Mathematics. Vol. 23. N 2. P. 263–278.
11. Ivanov I. and Nikitchenko M. (2018). On the sequence rule for the Floyd-Hoare logic with partial pre- and post-conditions. In Proceedings of the 14th International Conference on ICT. Vol. 2104 of CEUR Workshop Proc. P. 716–724.
12. Kleene S. (1973) Mathematical Logic. Moscow: Mir (in rus).
13. Kleene S. (1952) Introductions to Metamathematics. Van Nostrand, Princeton.
14. Nikitchenko M., Shkilniak O. and Shkilniak S. (2018). Logics of general non-deterministic predicates: semantic aspects. In Problems in Progamming. N 2–3. P. 31–45 (in ukr).

Одержано 07.02.2019

Про авторів:

Нікітченко Микола Степанович,
доктор фізико-математичних наук,
професор, завідувач кафедри Теорії
та технології програмування.
Кількість наукових публікацій в
українських виданнях – понад 250,
у тому числі у фахових виданнях –
понад 110.
Кількість наукових публікацій в
зарубіжних виданнях – понад 60.
Scopus Author ID: 6602842336.
h-індекс (Google Scholar): 13 (11 з 2014).
http://orcid.org/0000-0002-4078-1062.

Шкільняк Оксана Степанівна,
кандидат фізико-математичних наук,
доцент, доцент кафедри
інформаційних систем.
Кількість наукових публікацій в
українських виданнях – понад 90,
у тому числі у фахових виданнях – 35.
Кількість наукових публікацій в
зарубіжних виданнях – 12.
Scopus Author ID: 57190873266
h-індекс (Google Scholar): 5 (4 з 2014)
http://orcid.org/0000-0003-4139-2525.

Шкільняк Степан Степанович,
доктор фізико-математичних наук,
професор, професор кафедри Теорії
та технології програмування.
Кількість наукових публікацій в
українських виданнях – понад 240,
у тому числі у фахових виданнях –
понад 100.
Кількість наукових публікацій в
зарубіжних виданнях – 22.
Scopus Author ID: 36646762300
h-індекс (Google Scholar): 7 (5 з 2014).
http://orcid.org/0000-0001-8624-5778.

Мамедов Тогрул Алірзайович,
аспірант кафедри Теорії
та технології програмування.
Кількість наукових публікацій в
українських виданнях – 9.
http://orcid.org/0000-0002-6049-604X

***Місце роботи авторів*:**

Київський національний університет
імені Тараса Шевченка,
01601, Київ, вул. Володимирська, 60.
Тел.: (044) 259 05 19.
E-mail: me.oksana@gmail.com

УДК 004.42:510.69 https://doi.org/10.15407/pp2019.01.014

О.С. Шкільняк

ДЕВІАНТНІ АЛГЕБРИ ІСТИННІСНИХ ЗНАЧЕНЬ ТА ДЕВІАНТНІ КЛАСИ ЗАГАЛЬНИХ НЕДЕТЕРМІНОВАНИХ ПРЕДИКАТІВ

Вивчаються програмно-орієнтовані логічні формалізми – логіки загальних недетермінованих (*GND*) предикатів. *GND*-предикати моделюються як 7-значні *TD*7-предикати. Досліджено девіантні алгебри істиннісних значень (*TV*-алгебри) *TD*7-предикатів та девіантні класи *GND*-предикатів. Девіантна *TV*-алгебра не індукує алгебру *GND*-предикатів. Для підмножин істиннісних значень досліджено можливість модифікації $\vee_*$ із умовою коректності *TFC*, що визначає відповідні класи *GND*-предикатів. Описано природні модифікації $\vee_*$ без *TFC*, що дає низку девіантних *TV*-алгебр.

Ключові слова: логіка, алгебра, недетермінований предикат, 7-значний предикат.

Вступ

Основа сучасних інформаційних та програмних систем – це апарат математичної логіки. Різноманітні логічні формалізми успішно використовуються для розв'язання широкого кола задач інформатики й програмування [1 – 4]. Водночас розвиток та розширення сфери застосування інформаційних технологій зумовлює необхідність розробки нових логік, які більше адаптовані до потреб програмування й моделювання. Найперше, ці логіки мають враховувати широке використання в в програмних системах та системах штучного інтелекту часткових недетермінованих відображень над неповними даними. До таких програмно-орієнтовних логічних формалізмів належать композиційно-номінативні логіки (КНЛ) квазіарних предикатів [5 – 7].

Важливим класом КНЛ є логіки загальних недетермінованих квазіарних предикатів, або *GND*-предикатів. Ці логіки запропоновано в [8], вони вивчались в [8, 9]. Зазначені логіки відображають такі властивості програм, як частковість, недетермінізм, нефіксовану арність. В роботах [8, 9] виділено різновиди *GND*-предикатів, описано композиційні алгебри та мови логіки *GND*-предикатів. *GND*-предикати можна моделювати як 7-значні тотальні детерміновані (*TD*7) предикати. Описано усі 20 підалгебр алгебри істиннісних значень (*TV*-алгебри) *TD*7-предикатів $ATV_7 = (TV_7, \{\neg_*, \vee_*\})$. Досліджено індукування цими підалгебрами відповідних алгебр *TD*7-предикатів та *GND*-предикатів.

Існує надзвичайно багато 7-значних логік, тому багато підмножин TV_7 незамкнені щодо $\neg_*$ чи $\vee_*$, вони не утворюють підалгебр ATV_7. Такі підмножини та відповідні їм класи *GND*-предикатів названо *девіантними*. Дослідженню девіантних *TV*-алгебр та девіантних класів *GND*-предикатів присвячена дана робота.

Для того, щоб девіантна $TV \subseteq TV_7$ утворила алгебру, необхідно модифікувати $\neg_*$ чи $\vee_*$. Модифікація $\neg_*$ веде до специфічних некласичних логік, вона в роботі не розглядається. Найважливішими є модифікації $\vee_*$, для яких виконується умова *TFC* коректності логічних зв'язок предикатних алгебр. При порушенні *TFC* *TV*-алгебра *девіантна*, вона не індукує алгебру *GND*-предикатів. Для всіх підмножин TV_7 в роботі досліджено можливість модифікації $\vee_*$ із умовою *TFC*, що визначає відповідні класи *GND*-предикатів. Описано природні модифікації $\vee_*$ без *TFC*, що дає низку девіантних *TV*-алгебр. Для деяких девіантних множин не існує модифікацій $\vee_*$ із умовою *TFC*, для них вказано відносно природні девіантні *TV*-алгебри.

Поняття, які в цій роботі не визначаються, тлумачимо в сенсі [6, 8, 9].

1. *GND* та *TD*7-предикати

V-A-квазіарним предикатом називають часткову неоднозначну, взагалі кажучи, функцію вигляду $P : {}^{V}A \to \{T, F\}$. Тут $\{T, F\}$ – множина істиннісних значень, ${}^{V}A$ – множина всіх *V-A*-іменних множин.

Часткові неоднозначні предикати на множині ${}^{V}A$ можна трактувати як відповідності (відношення) між ${}^{V}A$ та $\{T, F\}$. Такі предикати названо [5, 6] квазіарними предикатами реляційного типу, або *R*-предикатами. Для *R*-предикатів неможлива ситуація, коли при застосуванні предиката *P* до певного даного *d* виробляється результат (можливо, не один), водночас застосування *P* до цього *d* може дати невизначеність (результат не виробляється).

В загальному випадку поняття неоднозначного (недетермінованого) квазіарного предиката адекватно уточнюється [8, 9] як поняття *GND*-предиката – загального недетермінованого предиката. Такий предикат $P : {}^{V}A \to \{T, F\}$ при застосуванні до даного $d \in {}^{V}A$ може приймати значення *T*, приймати значення *F*, а може і не приймати жодного значення (бути невизначеним). Множиною значень *P*[*d*], які *GND*-предикат *P* може прийняти на даному $d \in {}^{V}A$, може бути одна з множин $\{\varnothing\}$, $\{T\}$, $\{F\}$, $\{T, F\}$, $\{T, \varnothing\}$, $\{F, \varnothing\}$, $\{T, F, \varnothing\}$.

Скорочено позначимо ці множини як ↑, *T*, *F*, *TF*, *T*↑, *F*↑, *TF*↑.

Кожний *GND*-предикат *P* можна однозначно описати за допомогою 3-х множин: області істинності *T*(*P*), області хибності *F*(*P*) та області невизначеності ⊥(*P*). Ці множини задаються наступним чином:

– $T(P) = \{d \mid T \in P[d]\}$;

– $F(P) = \{d \mid F \in P[d]\}$;

– $\perp(P) = \{d \mid \varnothing \in P[d]\} =$

$= \{d \mid P$ може бути невизначеним на $d\}$.

Такі множини пов'язує умова:

$$F(P) \cup T(P) \cup \perp(P) = {}^{V}A \quad .$$

Кожний *R*-предикат $P : {}^{V}A \to \{T, F\}$ на даному $d \in {}^{V}A$ може приймати лише значення *T*, лише значення *F*, обидва значення *T* та *F*, та може бути невизначеним. Тому для *R*-предиката *P* множина *P*[*d*] може бути однією з $\{\varnothing\}$, $\{T\}$, $\{F\}$, $\{T, F\}$. Кожний *R*-предикат *P* можна однозначно задати за допомогою 2-х множин: *T*(*P*) та *F*(*P*). Тоді ⊥(*P*) є доповненням до $T(P) \cup F(P)$.

Накладаючи ті чи інші обмеження на області істинності, хибності та невизначеності, виділено [8, 9] низку класів *GND*-предикатів. Зокрема, маємо 7 константних *GND*-предикатів: ⊥, T, F, ϒ, $T_{↑}$, $F_{↑}$, $ϒ_{↑}$.

GND-предикати можна моделювати [8] як 7-значні тотальні детерміновані предикати, або *TD*7-предикати. Множиною істиннісних значень *TD*7-предикатів є $TV_7 = \{T, F, T↑, F↑, ↑, TF, TF↑\}$.

Класи *V-A*-квазіарних *GND*-предикатів та *TD*7-предикатів позначають відповідно $PrG_{V\text{–}A}$ та $PrTD7_{V\text{–}A}$.

Базові пропозиційні композиції *TD*7-предикатів – це логічні зв'язки заперечення $\neg_*$ та диз'юнкція $\vee_*$. Задаємо їх традиційним чином – за допомогою таблиць істинності (табл. 1, 2).

Таблиця 1. Композиція $\neg_*$

P	*T*	*F*	*T*↑	*F*↑	↑	*TF*	*TF*↑
$\neg_*$P	*F*	*T*	*F*↑	*T*↑	↑	*TF*	*TF*↑

Таблиця 2. Композиція $\vee_*$

$\vee_*$	*T*	*F*	*T*↑	*F*↑	↑	*TF*	*TF*↑
T	*T*	*T*	*T*	*T*	*T*	*T*	*T*
F	*T*	*F*	*T*↑	*F*↑	↑	*TF*	*TF*↑
T↑	*T*	*T*↑	*T*↑	*T*↑	*T*↑	*T*↑	*T*↑
F↑	*T*	*F*↑	*T*↑	*F*↑	↑	*TF*↑	*TF*↑
↑	*T*	↑	*T*↑	↑	↑	*T*↑	*T*↑
TF	*T*	*TF*	*T*↑	*TF*↑	*T*↑	*TF*	*TF*↑
TF↑	*T*	*TF*↑	*T*↑	*TF*↑	*T*↑	*TF*↑	*TF*↑

Композиція кон'юнкції $\&_*$ – похідна:

$$P \&_* Q = \neg_*(\neg_* P \vee_* \neg_* Q).$$

Тому $\&_*$ можна подати так (табл. 3).

Таблиця 3. Композиція $\&_*$

$\&_*$	T	F	$T\uparrow$	$F\uparrow$	$\uparrow$	TF	$TF\uparrow$
T	T	F	$T\uparrow$	$F\uparrow$	$\uparrow$	TF	$TF\uparrow$
F	F	F	F	F	F	F	F
$T\uparrow$	$T\uparrow$	F	$T\uparrow$	$F\uparrow$	$\uparrow$	$TF\uparrow$	$TF\uparrow$
$F\uparrow$	$F\uparrow$	F	$F\uparrow$	$F\uparrow$	$F\uparrow$	$F\uparrow$	$F\uparrow$
$\uparrow$	$\uparrow$	F	$\uparrow$	$F\uparrow$	$\uparrow$	$F\uparrow$	$F\uparrow$
TF	TF	F	$TF\uparrow$	$F\uparrow$	$F\uparrow$	TF	$TF\uparrow$
$TF\uparrow$	$TF\uparrow$	F	$TF\uparrow$	$F\uparrow$	$F\uparrow$	$TF\uparrow$	$TF\uparrow$

Множина істиннісних значень TV_7 замкнена щодо $\neg_*$, $\vee_*$ та $\&_*$.

2. Алгебра істиннісних значень *TD*7-предикатів та її підалгебри

Алгебру $ATV_7 = (TV_7, \{\neg_*, \vee_*\})$ назвемо *алгеброю істиннісних значень*, або *TV-алгеброю TD*7-предикатів. Розглядають також *TV*-алгебру *TD*7-предикатів розширеної сигнатури $ATV_{7E} = (TV_7, \{\neg_*, \vee_*, \&_*\})$.

Для логічних зв'язок *TD*7-предикатів маємо [8, 9] традиційні властивості:

– комутативність $\vee_*$ та $\&_*$;

– асоціативність $\vee_*$ та $\&_*$;

– ідемпотентність $\vee_*$ та $\&_*$;

– зняття подвійного заперечення;

– закон контрапозиції;

– закони де Моргана.

Проте для TD7-предикатів неправильні закони дистрибутивності $\vee*$ щодо $\&*$ та $\&*$ щодо $\vee*$, також неправильні закони поглинання.

Приклад 1.

$(\uparrow \vee_* T) \&_* TF = T \&_* TF = TF$, водночас

$(\uparrow \&_* TF) \vee_* (T \&_* TF) = F\uparrow \vee_* TF = TF\uparrow$;

$(\uparrow \&_* F) \vee_* TF = F \vee_* TF = TF$, водночас

$(\uparrow \vee_* TF) \&_* (F \vee_* TF) = TF\uparrow \&_* TF = TF\uparrow$.

Приклад 2.

$TF \vee_* (TF \&_* \uparrow) = TF \vee_* F\uparrow = TF\uparrow \neq TF$;

$TF \&_* (TF \vee_* \uparrow) = TF \&_* T\uparrow = TF\uparrow \neq TF$.

Це цілком зрозуміло в світлі того, що *GND*-предикати моделюються *TD*7-предикатамии, а в логіці *GND*-предикатів не виконуються [8] закони дистрибутивності та закони поглинання.

Для множини TV_7 задаємо [8] природне впорядкування щодо $\vee_*$ та щодо $\&_*$.

Впорядкування TV_7 щодо $\vee_*$:

$$\alpha \rightarrow_\vee \beta, \text{ якщо } \alpha \vee_* \beta = \beta.$$

Впорядкування TV_7 щодо $\&_*$:

$$\alpha \rightarrow_\& \beta, \text{ якщо } \alpha \&_* \beta = \alpha.$$

Твердження 1. Впорядкування $\rightarrow_\vee$ та $\rightarrow_\&$ транзитивні:

$$\alpha \rightarrow_\vee \beta \text{ та } \beta \rightarrow_\vee \gamma \Rightarrow \alpha \rightarrow_\vee \gamma.$$

$$\alpha \rightarrow_\& \beta \text{ та } \beta \rightarrow_\& \gamma \Rightarrow \alpha \rightarrow_\& \gamma.$$

Таку транзитивність гарантує асоціативність $\vee_*$ та $\&_*$:

$$\alpha \vee_* \gamma = \alpha \vee_* (\beta \vee_* \gamma) =$$

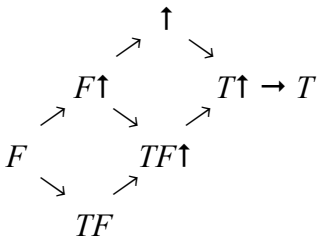
$$= (\alpha \vee_* \beta) \vee_* \gamma = \beta \vee_* \gamma = \gamma;$$

$$\alpha \&_* \gamma = \alpha \&_* (\beta \&_* \gamma) = (\alpha \&_* \beta) \&_* \gamma =$$

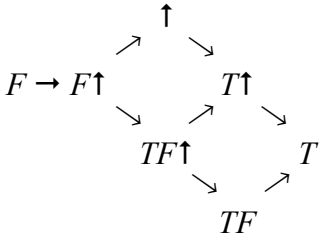
$$= \beta \&_* \gamma = \gamma.$$

Наведемо діаграми Хассе для множини TV_7 щодо $\vee_*$ та $\&_*$ [8].

Діаграма Хассе для TV_7 щодо $\vee_*$:



Діаграма Хассе для TV_7 щодо $\&_*$:



Це означає, що TV_7 не утворює решітки істиннісних значень. Причиною цього є те, що $\vee_*$ та $\&_*$ *невідмінні* на множині $\{TF, TF\uparrow\}$.

Для опису підалгебр ATV_7 виділено [8] всі 20 підмножин множини TV_7, які замкнені щодо $\neg_*$, $\vee_*$ та $\&_*$. Ці підмножини TV_{m_n} задають підалгебри ATV_{m_n} алгебри ATV_7. Далі ATV_{m_n} індукують підалгебри алгебри $ATD7_{V\text{–}A}$, які в свою чергу індукують підалгебри алгебри *GND*-предикатів відповідного рівня: пропозиційної алгебри $APG_{V\text{–}A}$, реномінативної $ARG_{V\text{–}A}$, першопорядкової $AQG_{V\text{–}A} = (PrG_{V\text{–}A}, \{\neg, \vee, \mathrm{R}^{\bar{v}}_{\bar{x}}, \exists x\})$. Зауважимо, що існує 30 підмножин TV_7, які замкнені лише щодо $\neg_*$.

Зв'язок ATV_7 та предикатних алгебр встановлюється так. Наявність істиннісного значення τ в TV_7 означає, що існують $P \in PrG_{V\text{–}A}$ та $d \in {}^{V}A$ такі: $\tau \in P[d]$.

Наведемо усі підмножини множини TV_7, замкнені щодо $\neg_*$, $\vee_*$ та $\&_*$, вказавши індуковані ними відповідні підалгебри ATV_{m_n} алгебр ATV_7 та підалгебри $AQG_{V\text{–}A}$.

$TV_{6\_1} = \{T, F, T\uparrow, F\uparrow, \uparrow, TF\uparrow\}$

індукує $AQAU_{V\text{–}A}$;

$TV_{6\_2} = \{T, F, T\uparrow, F\uparrow, TF, TF\uparrow\}$

індукує $AQTG_{V\text{–}A}$;

$TV_{5\_1} = \{T, F, T\uparrow, F\uparrow, \uparrow\}$

індукує $AQSG_{V\text{–}A}$;

$TV_{5\_2} = \{T, F, T\uparrow, F\uparrow, TF\uparrow\}$

індукує $AQTAU_{V\text{–}A}$;

$TV_{5\_3} = \{T\uparrow, F\uparrow, \uparrow, TF, TF\uparrow\}$

індукує $AQImG_{V\text{–}A}$;

$TV_{4\_1} = \{T, F, T\uparrow, F\uparrow\}$

індукує $AQTSG_{V\text{–}A}$;

$TV_{4\_2} = \{T, F, TF, TF\uparrow\}$

індукує $AQUA_{V\text{–}A}$;

$TV_{4\_3} = \{T\uparrow, F\uparrow, TF, TF\uparrow\}$

індукує $AQTImG_{V\text{–}A}$;

$TV_{4\_4} = \{T\uparrow, F\uparrow, \uparrow, TF\uparrow\}$

індукує $AQTIG_{V\text{–}A}$;

$TV_{3\_1} = \{T, F, \uparrow\}$

індукує $AQP_{V\text{–}A}$;

$TV_{3\_2} = \{T, F, TF\}$

індукує $AQT_{V\text{–}A}$;

$TV_{3\_3} = \{T, F, TF\uparrow\}$

індукує $AQU_{=A_{V\text{–}A}}$;

$TV_{3\_4} = \{T\uparrow, F\uparrow, TF\uparrow\}$

індукує $AQTTIG_{V\text{–}A}$;

$TV_{3\_5} = \{\uparrow, T\uparrow, F\uparrow\}$

індукує $AQSTIG_{V\text{–}A}$;

$TV_{2\_1} = \{T, F\}$

індукує $AQTS_{V\text{–}A}$;

$TV_{2\_2} = \{T\uparrow, F\uparrow\}$

індукує $AQTSTIG_{V\text{–}A}$;

$TV_{2\_3} = \{TF, TF\uparrow\}$

індукує $AQTAmG_{V\text{–}A}$;

$TV_{1\_1} = \{\uparrow\}$

індукує $AQ\bot_{V\text{–}A}$;

$TV_{1\_2} = \{TF\}$

індукує $AQ\Upsilon_{V\text{–}A}$;

$TV_{1\_3} = \{TF\uparrow\}$

індукує $AQ\Upsilon\bot_{V\text{–}A}$.

Необхідною умовою коректності ("природності") логічних зв'язок предикатних алгебр можна вважати умову *TFC* (*TF*-correct). Для $\vee$ та $\neg$вона задається так (тут $\alpha, \beta \in PrG_{V\text{–}A}$):

$$T(\alpha\vee\beta) = T(\alpha)\cup T(\beta),\ F(\alpha\vee\beta) = F(\alpha)\cap F(\beta);$$

$$T(\neg\alpha) = F(\alpha),\ F(\neg\alpha) = T(\alpha).$$

Виконання умови *TFC* має бути, зокрема, для константних предикатів T, F, $\bot$, $\mathrm{T}_\uparrow$, $\mathrm{F}_\uparrow$, Υ, $\Upsilon_\uparrow$, індукованих істиннісними значеннями $\uparrow$, T, F, TF, $T\uparrow$, $F\uparrow$, $TF\uparrow$.

Умова *TFC* виконується для алгебр $APG_{V\text{–}A}$, $ARG_{V\text{–}A}$, $AQG_{V\text{–}A}$, тому вона має виконуватися для всіх її підалгебр, індукованих відповідними підалгебрами ATV_7.

3. Девіантні підмножини TV_7 та індуковані ними логіки *GND*-предикатів

Задаючи ті чи інші умови для областей істинності, хибності та невизначеності, виділено низку різноманітних класів

GND-предикатів. Водночас не всі так описані класи замкнені щодо композиції диз'юнкції *GND*-предикатів. Це, зокрема, клас *R*-предикатів та класи *AnU*-предикатів, *TAnU*-предикатів, *RAU*-предикатів, *UAU*-предикатів, $nU_{=}A$-предикатів, $TnU_{=}A$-предикатів. Ці класи індуковано відповідними класами *TD*7-предикатів із такими множинами істиннісних значень:

– $TV_{AnU} = \{T, F, T\uparrow, F\uparrow, \uparrow, TF\}$;

– $TV_{TAnU} = \{T, F, T\uparrow, F\uparrow, TF\}$;

– $TV_{RAU} = TV_{D5} = \{T, F, \uparrow, TF, TF\uparrow\}$;

– $TV_{R} = \{T, F, \uparrow, TF\}$;

– $TV_{UAU} = \{T, F, \uparrow, TF\uparrow\}$;

– $TV_{nU=A} = \{T\uparrow, F\uparrow, \uparrow, TF\}$;

– $TV_{TnU=A} = \{T\uparrow, F\uparrow, TF\}$.

Ці множини замкнені щодо $\neg_*$, але незамкнені щодо $\vee_*$, тому вони не утворюють підалгебр ATV_7. Для того, щоб клас *TD*7-предикатів з такою множиною істиннісних значень утворював алгебру, треба тим чи іншим способом *модифікувати* $\vee_*$. Відповідно модифікується похідна $\&_*$.

Множини істиннісних значень, незамкнені щодо $\neg_*$ чи $\vee_*$, назвемо *девіантними*. Класи *TD*7-предикатів із девіантними множинами істиннісних значень та відповідні класи *GND*-предикатів теж назвемо *девіантними*. Девіантні класи *GND*-предикатів незамкнені щодо $\neg$ чи $\vee$.

Ми не розглядатимемо можливість модифікації композиції $\neg_*$, така модифікація веде до цілком некласичних логік, які вимагають окремого дослідження. З цієї ж причини розглядаємо такі модифікації $\vee_*$, які можуть мати відмінні від $\vee_*$ значення тільки для аргументів $\uparrow$, *TF*, $TF\uparrow$.

Отримані при модифікації $\vee_*$ *TV*-алгебри індукують алгебри *TD*7-предикатів, які далі індукують відповідні алгебри *GND*-предикатів на пропозиційному, реномінативному, першопорядковому рівнях. В цих алгебрах відповідним чином модифікується $\vee$, тому вони не будуть підалгебрами алгебр APG_{V-A}, ARG_{V-A}, AQG_{V-A}. В деяких випадках маємо *ізоморфізм* цих алгебр та певних підалгебр алгебр APG_{V-A}, ARG_{V-A}, AQG_{V-A}, а тому маємо їх *вкладення* в APG_{V-A}, ARG_{V-A}, AQG_{V-A}.

Нехай $TV \subseteq TV_7$. Так як $\neg_*$ не модифікуємо, то розглядаємо лише такі *TV*, які *замкнені* щодо $\neg_*$. Нехай в обмеженні таблиці істинності $\vee_*$ для *TV* маємо деяке істиннісне значення $\nu \notin TV$. Модифікація $\vee_*$ полягає у заміні значень ν в таблиці $\vee_*$ для *TV* на деяке $\tau \in TV$. Зрозуміло, що існує дуже багато способів це зробити. В першу чергу розглядаємо такі модифікації, для яких виконується умова *TFC* коректності логічних зв'язок предикатних алгебр. При порушенні *TFC* *TV*-алгебра *не індукує* для *GND*-предикатів композиційну алгебру із коректними пропозиційними композиціями. Таку *TV*-алгебру назвемо *девіантною*.

Для множин TV_{AnU}, TV_{TAnU}, TV_{D5}, TV_{R}, TV_{UAU}, $TV_{TnU=A}$ існують модифікації $\vee_*$ та $\vee$ із дотриманням умови *TFC*, тому й маємо такі класи *GND*-предикатів, як *AnU*, *TAnU*, *RAU*, *R*, *UAU*, $nU_{=}A$, $TnU_{=}A$.

Ще залишається розглянути множину істиннісних значень $\{\uparrow, TF, TF\uparrow\}$, яка незамкнена щодо $\vee_*$:

$$TF \vee_* \uparrow = T\uparrow \quad \text{та} \quad TF\uparrow \vee_* \uparrow = T\uparrow.$$

Її підмножини $\{TF, \uparrow\}$ та $\{TF\uparrow, \uparrow\}$ теж незамкнені щодо $\vee_*$. Далі покажемо, що для цих множин не існує модифікацій $\vee_*$, які індукують виконання *TFC*. Тому не існує *TV*-алгебр з носіями $\{\uparrow, TF, TF\uparrow\}$, $\{TF, \uparrow\}$ чи $\{TF\uparrow, \uparrow\}$, які індукують для *GND*-предикатів композиційну алгебру із коректними $\vee$ та $\&$. Інакше кажучи, усі такі *TV*-алгебри девіантні.

Логіка *AnU*-предикатів. Множина $TV_{AnU} = \{T, F, T\uparrow, F\uparrow, \uparrow, TF\}$ незамкнена щодо $\vee_*$: $TF \vee_* F\uparrow = TF\uparrow$.

Природна модифікація $\vee_*$ як $\vee_{AnU}$ така: $TF \vee_{AnU} F\uparrow = TF$.

Тоді TV_{AnU} замкнена щодо визначених нижче $\vee_{AnU}$ та $\&_{AnU}$ (табл. 4, 5).

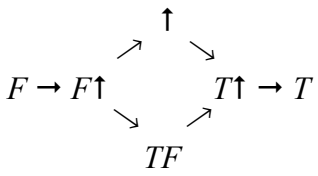
Таблиця 4. Композиція $\vee_{AnU}$

$\vee_{AnU}$	T	F	$T\uparrow$	$F\uparrow$	$\uparrow$	TF
T	T	T	T	T	T	T
F	T	F	$T\uparrow$	$F\uparrow$	$\uparrow$	TF
$T\uparrow$	T	$T\uparrow$	$T\uparrow$	$T\uparrow$	$T\uparrow$	$T\uparrow$
$F\uparrow$	T	$F\uparrow$	$T\uparrow$	$F\uparrow$	$\uparrow$	TF
$\uparrow$	T	$\uparrow$	$T\uparrow$	$\uparrow$	$\uparrow$	$T\uparrow$
TF	T	TF	$T\uparrow$	TF	$T\uparrow$	TF

Таблиця 5. Композиція $\&_{AnU}$

$\&_{AnU}$	T	F	$T\uparrow$	$F\uparrow$	$\uparrow$	TF
T	T	F	$T\uparrow$	$F\uparrow$	$\uparrow$	TF
F	F	F	F	F	F	F
$T\uparrow$	$T\uparrow$	F	$T\uparrow$	$F\uparrow$	$\uparrow$	TF
$F\uparrow$	$F\uparrow$	F	$F\uparrow$	$F\uparrow$	$F\uparrow$	$F\uparrow$
$\uparrow$	$\uparrow$	F	$\uparrow$	$F\uparrow$	$\uparrow$	$F\uparrow$
TF	TF	F	TF	$F\uparrow$	$F\uparrow$	TF

Діаграма Хассе для TV_{AnU} щодо $\vee_{AnU}$ та $\&_{AnU}$ така:



Отримуємо TV-алгебру ATV_{AnU}, яка індукує алгебру AnU-предикатів $AQAnU^{V-A}$.

Твердження 2. Модифікація $\vee_*$ як $\vee_{AnU}$ – єдина для TV_{AnU}, яка гарантує TFC.

Маємо

$T(TF \vee F\uparrow) = T(TF) \cup T(F\uparrow) = {}^{V}A \cup \varnothing = {}^{V}A$,

$F(TF \vee F\uparrow) = F(TF) \cap F(F\uparrow) = {}^{V}A \cap {}^{V}A = {}^{V}A$,

Отже, згідно TFC для $TF \vee_{AnU} F\uparrow$ можливі лише TF чи $TF\uparrow$.

Проте $TF\uparrow \notin TV_{AnU}$, тому TF – єдина можливість:

$T(TF \vee F\uparrow) = T(TF)$, $F(TF \vee F\uparrow) = F(TF)$.

Таблицю істинності $\vee_{AnU}$ для ATV_{AnU} можна отримати із таблиці істинності $\vee_*$ для ATV_{6_1} заміною $TF\uparrow$ на TF. Звідси

Твердження 3. $ATV_{AnU} \sim_{iz} ATV_{6\_1}$.

Наслідок 1. $AQAnU_{V-A} \sim_{iz} AQAU_{V-A}$.

Логіка *TAnU*-предикатів. Множина $TV_{TAnU} = \{T, F, T\uparrow, F\uparrow, TF\}$ незамкнена щодо $\vee_*$: $TF \vee_* F\uparrow = TF\uparrow$.

Маємо $TV_{TAnU} \subseteq TV_{AnU}$. Множина TV_{TAnU} замкнена щодо $\vee_{AnU}$ та $\&_{AnU}$.

Діаграма Хассе для TV_{TAnU} щодо $\vee_{AnU}$ та $\&_{AnU}$ така:

$$F \to F\uparrow \to TF \to T\uparrow \to T.$$

Маємо TV-алгебру ATV_{AnU}, яка індукує алгебру $TAnU$-предикатів $AQTAnU^{V-A}$.

Модифікація $\vee_*$ як $\vee_{AnU}$ – єдина для TV_{AnU}, яка гарантує умови TFC, тому така модифікація теж єдина, яка гарантує умови TFC, для її підмножини $TV_{TAnU} \subseteq TV_{AnU}$.

Таблицю істинності $\vee_{AnU}$ для ATV_{TAnU} можна отримати із таблиці істинності $\vee_*$ для ATV_{5_1} заміною $\uparrow$ на TF та із таблиці істинності $\vee_*$ для ATV_{5_2} заміною $TF\uparrow$ на TF. Звідси

Твердження 4.

$$ATV_{TAnU} \sim_{iz} ATV_{5\_1} \sim_{iz} ATV_{5\_2}.$$

Наслідок 2.

$$AQTAnU_{V-A} \sim_{iz} AQSG_{V-A} \sim_{iz} AQTAU_{V-A}.$$

Логіка *RAU*-предикатів. Множина $TV_{D5} = \{T, F, \uparrow, TF, TF\uparrow\}$ *незамкнена* щодо $\vee_*$: $TF \vee_* \uparrow = T\uparrow$; $TF\uparrow \vee_* \uparrow = T\uparrow$. Тому TV_{D5} не утворює підалгебри алгебри ATV_7.

Розглянемо допустимі варіанти модифікації $\vee_*$ у випадку TV_{D5}. Потрібно модифікувати значення $TF \vee_* \uparrow$ та $TF\uparrow \vee_* \uparrow$ так, щоб отримати елемент $\tau \in TV_{D5}$.

Маємо

$T(TF \vee \uparrow) = T(TF) \cup T(\uparrow) = T(TF) = {}^{V}A$ та $T(TF\uparrow \vee \uparrow) = T(TF\uparrow) \cup T(\uparrow) = T(TF\uparrow) = {}^{V}A$;

$F(TF \vee \uparrow) = F(TF) \cap F(\uparrow) = F(TF) \cap \varnothing = \varnothing$ та $F(TF\uparrow \vee \uparrow) = F(TF\uparrow) \cap F(\uparrow) = \varnothing$.

Отже, за умови TFC таким τ може бути лише T. Отримуємо

Твердження 5. Єдиною модифікацією $\vee_*$ для TV_{D5} за умови TFC є така $\vee_{RAU}$:

$TF \vee_{RAU} \uparrow = T$ та $TF\uparrow \vee_{RAU} \uparrow = T$.

Маємо *TV*-алгебру ATV_{RAU}, задамо її логічні зв'язки $\vee_{RAU}$ та $\&_{RAU}$ (табл. 6, 7).

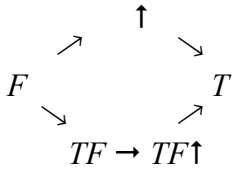
Таблиця 6. Композиція $\vee_{RAU}$

$\vee_{RAU}$	T	F	$\uparrow$	TF	$TF\uparrow$
T	T	T	T	T	T
F	T	F	$\uparrow$	TF	$TF\uparrow$
$\uparrow$	T	$\uparrow$	$\uparrow$	T	T
TF	T	TF	T	TF	$TF\uparrow$
$TF\uparrow$	T	$TF\uparrow$	T	$TF\uparrow$	$TF\uparrow$

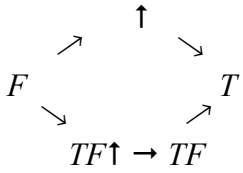
Таблиця 7. Композиція $\&_{RAU}$

$\&_{RAU}$	T	F	$\uparrow$	TF	$TF\uparrow$
T	T	F	$\uparrow$	TF	$TF\uparrow$
F	F	F	F	F	F
$\uparrow$	$\uparrow$	F	$\uparrow$	F	F
TF	TF	F	F	TF	$TF\uparrow$
$TF\uparrow$	$TF\uparrow$	F	F	$TF\uparrow$	$TF\uparrow$

Діаграма Хассе щодо $\vee_{RAU}$ така:



Діаграма Хассе щодо $\&_{RAU}$ така:



Твердження 6. ATV_{RAU} неізоморфна алгебрам ATV_{5_1}, ATV_{5_2}, ATV_{5_3}, ATV_{AnU}.

Наслідок 3. Алгебра $AQRAU_{V\text{–}A}$ неізоморфна алгебрам $AQSG_{V\text{–}A}$, $AQTAU_{V\text{–}A}$, $AQImG_{V\text{–}A}$, $AQAnU_{V\text{–}A}$.

Логіка *R*-предикатів. Особливе місце посідає множина істиннісних значень $TV_R = \{T, F, \uparrow, TF\}$, безпосередньо пов'язана з логікою *R*-предикатів.

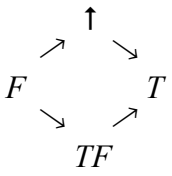
Множина TV_R незамкнена щодо $\vee_*$: $TF \vee_* \uparrow = T\uparrow$. Отже, TV_R не утворює підалгебри алгебри ATV_7.

Таким чином, алгебра *R*-предикатів $AQR_{V\text{–}A}$ не є підалгеброю алгебри $AQG_{V\text{–}A}$.

Маємо $TV_R \subseteq TV_{RAU}$. Множина TV_R замкнена щодо композицій $\vee_{RAU}$ та $\&_{RAU}$.

Алгебра ATV_R – це фактично алгебра ATV_B істиннісних значень відомої [10] 4-значної логіки Белнапа. Композиції $\vee_B$ та $\&_B$ логіки Белнапа – це звуження композицій $\vee_{RAU}$ та $\&_{RAU}$ на множину TV_R. Це гарантує виконання умови *TFC*.

Діаграма Хассе для множини TV_R щодо композицій $\vee_B$ та $\&_B$ така:



Твердження 7.

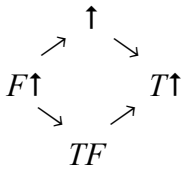
$ATV_{4\_4} \sim_{iz} ATV_B \sim_{iz} ATV_R$.

Алгебра $AQR_{V\text{–}A}$ є *вкладенням* в алгебри $AQImG_{V\text{–}A}$ та $AQAU_{V\text{–}A}$, тому й в $AQG_{V\text{–}A}$. Алгебра $AQR_{V\text{–}A}$ не індукується підалгебрами алгебри ATV_7. В $AQR_{V\text{–}A}$ композиція $\vee$ узгоджена із композицією $\vee_B$.

Наслідок 4. $AQR_{V\text{–}A} \sim_{iz} AQTIG_{V\text{–}A}$.

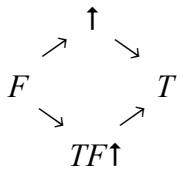
Логіка *nU=A*-предикатів. Множина $TV_{nU=A} = \{T\uparrow, F\uparrow, \uparrow, TF\} \subseteq TV_{AnU}$ незамкнена щодо $\vee_*$: $TF \vee_* F\uparrow = TF\uparrow$. Така $TV_{nU=A}$ замкнена щодо $\vee_{AnU}$ та $\&_{AnU}$. *TV*-алгебра $ATV_{nU=A}$ індукує алгебру $nU_{=}A$-предикатів $AQnU_{=}A_{V\text{–}A}$.

Діаграма Хассе для $TV_{nU=A}$ щодо $\vee_{AnU}$ та $\&_{AnU}$ така:



Логіка *UAU*-предикатів. Множина $TV_{UAU} = \{T, F, \uparrow, TF\uparrow\}$ незамкнена щодо $\vee_*$: $\uparrow \vee_* TF\uparrow = T\uparrow$. Маємо $TV_{UAU} \subseteq TV_{RAU}$, така TV_{UAU} замкнена щодо $\vee_{RAU}$ та $\&_{RAU}$. *TV*-алгебра ATV_{UAU} індукує алгебру *UAU*-предикатів $AQUAU_{V\text{–}A}$.

Діаграма Хассе для TV_{UAU} щодо $\vee_{RAU}$ та $\&_{RAU}$ така:



Порівнюючи таблиці істинності $\vee_{RAU}$ для ATV_{UAU}, $\vee_{AnU}$ для $ATV_{nU=A}$, $\vee_{RAU}$ для ATV_R та $\vee_*$ для ATV_{4_4}, отримуємо:

Твердження 8. $ATV_{UAU} \sim_{iz}$ $\sim_{iz} ATV_{nU=A} \sim_{iz} ATV_R \sim_{iz} ATV_{4\_4}$.

Наслідок 5. $AQUAU_{V-A} \sim_{iz}$ $\sim_{iz} AQnU_{=}A_{V-A} \sim_{iz} AQR_{V-A} \sim_{iz} AQTIG_{V-A}$.

Логіка *TnU=A*-предикатів. Множина $TV_{TnU=A} = \{T\uparrow, F\uparrow, TF\}$ незамкнена щодо $\vee_*$: $TF \vee_* F\uparrow = TF\uparrow$. Водночас $TV_{TnU=A}$ замкнена щодо $\vee_{AnU}$ та $\&_{AnU}$. *TV*-алгебра $ATV_{TnU=A}$ індукує алгебру $TnU_{=}A$-предикатів $AQTnU_{=}A_{V-A}$.

Діаграма Хассе для $TV_{TnU=A}$ щодо $\vee_{AnU}$ та $\&_{AnU}$ така:

$$F\uparrow \rightarrow TF \rightarrow T\uparrow$$

Порівнюючи таблицю істинності $\vee_{AnU}$ для $ATV_{TnU=A}$ із таблицями істинності $\vee_*$ для ATV_{3_1}, ATV_{3_2}, ATV_{3_3}, ATV_{3_4}, ATV_{3_5}, отримуємо:

Твердження 9. $ATV_{TnU=A} \sim_{iz} ATV_{3\_1}$ $\sim_{iz} ATV_{3\_2} \sim_{iz} ATV_{3\_3} \sim_{iz} ATV_{3\_4} \sim_{iz} ATV_{3\_5}$.

Наслідок 6.

$AQTnU_{=}A_{V-A} \sim_{iz} AQP_{V-A} \sim_{iz} AQT_{V-A} \sim_{iz}$ $\sim_{iz} AQTU_{=}A_{V-A} \sim_{iz} AQTTIG_{V-A} \sim_{iz} AQSTIG_{V-A}$.

4. Логіки, індуковані TV_{D5}

Єдиним допустимим варіантом модифікації $\vee_*$ за умови *TFC* у випадку TV_{D5} є такий: $TF \vee_{RAU} \uparrow = T$ та $TF\uparrow \vee_{RAU} \uparrow = T$. Він веде до логіки *RAU*-предикатів.

Інші варіанти модифікації $\vee_*$для TV_{D5} шляхом корекції виділених значень $T\uparrow$ в таблиці $\vee_*$ ведуть до *істотно* девіантних логік, для яких не виконується *TFC*.

Через порушення умови *TFC* *TV*-алгебри девіантні, вони не індукують для *GND*-предикатів композиційні алгебри із коректними $\vee$ та $\&$.

Розглянемо для TV_{D5} варіант модифікації $\vee_*$, який веде до специфічної логіки *EU*-предикатів, істотно відмінної від логіки *GND*-предикатів.

5-значна логіка *EU*. Оригінальна 5-значна логіка *EU* описана в [11]. Множиною істиннісних значень такої логіки є $TV_{EU} = \{T, F, \text{u}, \text{e}, \text{eu}\}$; при цьому u (невизначеність) трактується як ↑ у нас, проте трактування e (помилка) та eu дещо відмінне від їх трактування як *TF* та *TF*↑ в алгебрі *ATV7*. Проте для уніфікації позначень множиною істиннісних значень логіки *EU* вважаємо $TV_{D5} = \{T, F, \uparrow, TF, TF\uparrow\}$.

Логічна зв'язка $\neg_{\text{eu}}$ логіка *EU* ідентична логічній зв'язці $\neg_*$.

Логічні зв'язки $\vee_{\text{eu}}$ та $\&_{\text{eu}}$ логіки *EU* задаються так (табл. 8, 9).

Таблиця 8. Композиція $\vee_{\text{eu}}$

$\vee_{\text{eu}}$	*T*	*F*	↑	*TF*	*TF*↑
T	*T*	*T*	*T*	*T*	*T*
F	*T*	*F*	↑	*TF*	*TF*↑
↑	*T*	↑	↑	*TF*↑	*TF*↑
TF	*T*	*TF*	*TF*↑	*TF*	*TF*↑
TF↑	*T*	*TF*↑	*TF*↑	*TF*↑	*TF*↑

Таблиця 9. Композиція $\&_{\text{eu}}$

$\&_{\text{eu}}$	*T*	*F*	↑	*TF*	*TF*↑
T	*T*	*F*	↑	*TF*	*TF*↑
F	*F*	*F*	*F*	*F*	*F*
↑	↑	*F*	↑	*TF*↑	*TF*↑
TF	*TF*	*F*	*TF*↑	*TF*	*TF*↑
TF↑	*TF*↑	*F*	*TF*↑	*TF*↑	*TF*↑

Отже, $\vee_{\text{eu}}$ отримано такою модифікацією $\vee_*$:

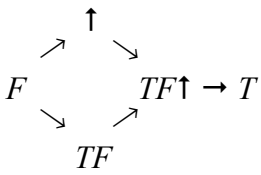
$TF \vee_{\text{eu}} \uparrow = TF\uparrow$ та $TF\uparrow \vee_{\text{eu}} \uparrow = TF\uparrow$.

Для *EU* умова *TFC не виконується*:

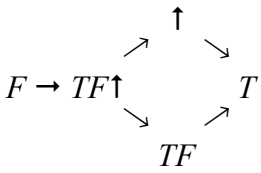
$F(TF\uparrow\vee\uparrow) = F(TF\uparrow)\cap F(\uparrow) = F(TF\uparrow)\cap\varnothing = \varnothing$

та $F(TF\vee\uparrow) = F(TF)\cap F(\uparrow) = F(TF)\cap\varnothing = \varnothing$, водночас $F(TF\uparrow) \neq \varnothing$.

Діаграма Хассе для TV_{D5} щодо $\vee_{eu}$:



Діаграма Хассе для TV_{D5} щодо $\&_{eu}$:



Маємо *TV*-алгебру *AEU*. Через порушення *TFC* алгебра *AEU* девіантна, вона не індукує для *GND*-предикатів композиційну алгебру із коректними $\vee$ та $\&$.

Алгебра *AEU* неізоморфна алгебрам ATV_{5_1}, ATV_{5_2}, ATV_{5_3}, ATV_{AnU}, ATV_{RAU}.

Опишемо ще 3 варіанти модифікації $\vee_*$ для TV_{D5}. Всі інші варіанти такої модифікації видаються зовсім неприродними.

Варіант_1. Модифікуємо $\vee_*$ як $\vee_{\#}$ так: $TF\vee_{\#}\uparrow = TF$ та $TF\uparrow\vee_{\#}\uparrow = TF\uparrow$.

Єдина відмінність $\vee_{\#}$ від $\vee_{eu}$ у тому, що $TF\vee_{eu}\uparrow = TF\uparrow$.

Умова *TFC не виконується*:

$F(TF\vee\uparrow) = F(TF)\cap F(\uparrow) = F(TF)\cap\varnothing = \varnothing$, водночас $F(TF) \neq \varnothing$.

Діаграма Хассе для TV_{D5} щодо $\vee_{\#}$:

$$F \to \uparrow \to TF \to TF\uparrow \to T$$

Діаграма Хассе для TV_{D5} щодо $\&_{\#}$:

$$F \to TF\uparrow \to TF \to \uparrow \to T$$

Маємо *TV*-алгебру, яку через подібність до *AEU* назвемо $AEU_{\#}$.

$AEU_{\#}$ неізоморфна алгебрам ATV_{5_1}, ATV_{5_2}, ATV_{5_3}, ATV_{RAU}, ATV_{TAnU}, *AEU*.

Через порушення *TFC* девіантна алгебра *AEU#* не індукує композиційну алгебру *GND*-предикатів із коректними $\vee$ та $\&$.

Варіант_2. Модифікуємо $\vee_*$ як $\vee_{\#D}$ так: $TF\vee_{\#D}\uparrow = TF$ та $TF\uparrow\vee_{\#D}\uparrow = TF$.

Отримуємо логічні зв'язки $\vee_{\#D}$ та $\&_{\#D}$, що задаються так (табл. 10, 11).

Таблиця 10. Композиція $\vee_{\#D}$

$\vee_{\#D}$	T	F	$\uparrow$	TF	$TF\uparrow$
T	T	T	T	T	T
F	T	F	$\uparrow$	TF	$TF\uparrow$
$\uparrow$	T	$\uparrow$	$\uparrow$	TF	TF
TF	T	TF	TF	TF	$TF\uparrow$
$TF\uparrow$	T	$TF\uparrow$	TF	$TF\uparrow$	$TF\uparrow$

Таблиця 11. Композиція $\&_{\#D}$

$\&_{\#D}$	T	F	$\uparrow$	TF	$TF\uparrow$
T	T	F	$\uparrow$	TF	$TF\uparrow$
F	F	F	F	F	F
$\uparrow$	$\uparrow$	F	$\uparrow$	TF	TF
TF	TF	F	TF	TF	$TF\uparrow$
$TF\uparrow$	$TF\uparrow$	F	TF	$TF\uparrow$	$TF\uparrow$

Умова *TFC не виконується*:

$F(TF\vee\uparrow) = F(TF)\cap F(\uparrow) = F(TF)\cap\varnothing = \varnothing$, водночас $F(TF) \neq \varnothing$.

$\vee_{\#D}$ і $\&_{\#D}$ *неасоціативні.* Покажемо для $\vee_{\#D}$: $\uparrow\vee_{\#D}(TF\uparrow\vee_{\#D}TF) = \uparrow\vee_{\#D}TF\uparrow = TF$; $(\uparrow\vee_{\#D}TF)\vee_{\#D}TF\uparrow = TF\vee_{\#D}TF\uparrow = TF\uparrow$.

$\vee_{\#D}$ і $\&_{\#D}$ *не задають* впорядкування TV_{D5} через порушення транзитивності. Показуємо це для $\vee_{\#D}$. $\alpha \to \beta$ означає $\alpha\vee_{\#D}\beta = \beta$, тому $\uparrow \to TF$ та $TF \to TF\uparrow$. За транзитивністю має бути $\uparrow \to TF\uparrow$, тобто $\uparrow\vee_{\#D}TF\uparrow = TF\uparrow$. Проте $\uparrow\vee_{\#D}TF\uparrow = TF$.

Отже, отримана *TV*-алгебра $AEU_{\#D}$ ***цілком девіантна***.

Варіант_3. Маємо $TV_R \subseteq TV_{D5}$, тому розглянемо модифікацію $\vee_{\#T}$ композиції $\vee_*$, яка є розширенням $\vee_B$ на TV_{D5}.

Модифікуємо $\vee_*$ як $\vee_{\#T}$ так:

$$TF\vee_{\#T}\uparrow = T \text{ та } TF\uparrow\vee_{\#T}\uparrow = TF\uparrow.$$

Така $\vee_{\#T}$ відмінна від $\vee_{RAU}$ лише у тому, що $TF\uparrow\vee_{RAU}\uparrow = T$.

Отримуємо логічні зв'язки $\vee_{\#T}$ та $\&_{\#T}$, що задаються так (табл. 12, 13).

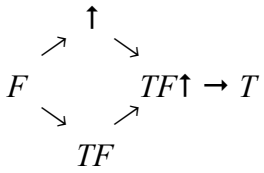
Таблиця 12. Композиція $\vee_{\#T}$

$\vee_{\#T}$	T	F	$\uparrow$	TF	$TF\uparrow$
T	T	T	T	T	T
F	T	F	$\uparrow$	TF	$TF\uparrow$
$\uparrow$	T	$\uparrow$	$\uparrow$	T	$TF\uparrow$
TF	T	TF	T	TF	$TF\uparrow$
$TF\uparrow$	T	$TF\uparrow$	$TF\uparrow$	$TF\uparrow$	$TF\uparrow$

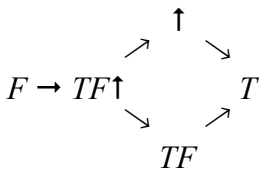
Таблиця 13. Композиція $\&_{\#T}$

$\&_{\#T}$	T	F	$\uparrow$	TF	$TF\uparrow$
T	T	F	$\uparrow$	TF	$TF\uparrow$
F	F	F	F	F	F
$\uparrow$	$\uparrow$	F	$\uparrow$	F	$TF\uparrow$
TF	TF	F	F	TF	$TF\uparrow$
$TF\uparrow$	$TF\uparrow$	F	$TF\uparrow$	$TF\uparrow$	$TF\uparrow$

Діаграма Хассе для TV_{D5} щодо $\vee_{\#T}$:



Діаграма Хассе для TV_{D5} щодо $\&_{\#T}$:



TFC не виконується: $F(TF\uparrow\vee\uparrow) = F(TF\uparrow)\cap F(\uparrow) = F(TF\uparrow)\cap\varnothing = \varnothing \neq F(TF\uparrow)$.

$\vee_{\#T}$ і $\&_{\#T}$ *неасоціативні.* Покажемо для $\vee_{\#T}$: $\uparrow \vee_{\#T} (TF\uparrow \vee_{\#T} TF) = \uparrow \vee_{\#T} TF\uparrow = TF\uparrow$; $(\uparrow \vee_{\#T} TF) \vee_{\#T} TF\uparrow = T \vee_{\#T} TF\uparrow = T$.

Отримали *цілком девіантну* *TV*-алгебру, яку назвемо AEU_T.

AEU_T неізоморфна алгебрам ATV_{5_1}, ATV_{5_2}, ATV_{5_3}, ATV_{RAU}, ATV_{TAnU}, AEU, $AEU_{\#}$.

TV*-алгебри, індуковані підмножинами TV_{D5}**. Маємо три **4-*елементних підмножини TV_{D5}, які замкнені щодо $\neg_*$.

Це TV_{4_2}, TV_R та TV_{UAU}.

Множина $TV_{4\_2} = \{T, F, TF, TF\uparrow\}$ генерує *TV*-алгебру ATV_{4_2}, яка індукує алгебри *TUA*-предикатів.

Множина $TV_R = \{T, F, \uparrow, TF\}$ незамкнена щодо $\vee_*$: $TF \vee_* \uparrow = T\uparrow$.

Модифікація $\vee_*$ як $\vee_{RAU}$ дає *TV*-алгебру ATV_R, яка індукує алгебри *R*-предикатів.

Множина $TV_{UAU} = \{T, F, \uparrow, TF\uparrow\}$ незамкнена щодо $\vee_*$: $\uparrow \vee_* TF\uparrow = T\uparrow$.

Модифікація $\vee_*$ як $\vee_{RAU}$ дає *TV*-алгебру ATV_{UAU}, яка індукує алгебри *UAU*-предикатів.

Модифікація $\vee_*$ як $\vee_{eu}$ дає *TV*-алгебру AEU_{4_1}.

Зауважимо, що на множині TV_{UAU} із $\vee_{eu}$ та $\&_{eu}$ збігаються $\vee_{\#}$ та $\&_{\#}$, $\vee_{\#T}$ та $\&_{\#T}$.

Діаграма Хассе для EU_{4_1} щодо $\vee_{eu}$:

$$F \rightarrow \uparrow \rightarrow TF\uparrow \rightarrow T.$$

Діаграма Хассе для EU_{4_1} щодо $\&_{eu}$:

$$F \rightarrow TF\uparrow \rightarrow \uparrow \rightarrow T.$$

Маємо *TV*-алгебру AEU_{4_1}.

Для AEU_{4_1} *TFC не виконується*:

$$F(TF\uparrow\vee\uparrow) = F(TF\uparrow)\cap F(\uparrow) = F(TF\uparrow)\cap\varnothing = \varnothing,$$

водночас $F(TF\uparrow) \neq \varnothing$. Тому девіантна AEU_{4_1} *не індукує* композиційну алгебру *GND*-предикатів із коректними $\vee$ та $\&$.

AEU_{4_1} – це алгебра ATV_{4_2}, в якій $TF \Rightarrow \uparrow$. Тому $AEU_{4\_1} \sim_{iz} ATV_{4\_2}$. Водночас AEU_{4_1} неізоморфна алгебрам ATV_{4_1}, ATV_{4_3}, ATV_{4_4}, $ATV_{nU=A}$, ATV_R, ATV_{UAU}.

Розглянемо ще одну модифікацію $\vee_*$ на TV_{UAU} – це $\vee_{42}$ така: $TF\uparrow \vee_{42} \uparrow = \uparrow$.

Така $\vee_{42}$ та похідна $\&_{42}$ видаються неприродними (табл. 14, 15).

Таблиця 14. Композиція $\vee_{42}$

$\vee_{42}$	T	F	$\uparrow$	$TF\uparrow$
T	T	T	T	T
F	T	F	$\uparrow$	$TF\uparrow$
$\uparrow$	T	$\uparrow$	$\uparrow$	$\uparrow$
$TF\uparrow$	T	$TF\uparrow$	$\uparrow$	$TF\uparrow$

Таблиця 15. Композиція $\&_{42}$

$\&_{42}$	T	F	$\uparrow$	$TF\uparrow$
T	T	F	$\uparrow$	$TF\uparrow$
F	F	F	F	F
$\uparrow$	$\uparrow$	F	$\uparrow$	$\uparrow$
$TF\uparrow$	$TF\uparrow$	F	$\uparrow$	$TF\uparrow$

Модифікація $\vee_*$ як $\vee_{42}$ дає *TV*-алгебру AEU_{4_2}.

Для AEU_{4_2} *TFC не виконується:* $T(TF\uparrow\vee\uparrow) = T(TF\uparrow)\cup T(\uparrow) = T(TF\uparrow)\cup\varnothing = T(TF\uparrow) \neq \varnothing$, водночас $T(\uparrow) = \varnothing$.

Діаграма Хассе щодо $\vee_{42}$ така:

$$F \to TF\uparrow \to \uparrow \to T.$$

Діаграма Хассе щодо $\&_{42}$ така:

$$F \to \uparrow \to TF\uparrow \to T.$$

AEU_{4_2} – це ATV_{4_2}, в якій $TF\uparrow \Rightarrow \uparrow$ та $TF \Rightarrow TF\uparrow$. Тому $AEU_{4\_2} \sim_{iz} ATV_{4\_2}$.

Отже, $AEU_{4\_1} \sim_{iz} AEU_{4\_2} \sim_{iz} ATV_{4\_2}$.

Через порушення *TFC* девіантна AEU_{4_2} не індукує композиційну алгебру *GND*-предикатів із коректними $\vee$ та $\&$.

Маємо чотири 3-***елементних*** підмножини TV_{D5}, які замкнені щодо $\neg_*$. Три з них замкнені щодо $\vee_*$.

Множина $TV_{3\_1} = \{T, F, \uparrow\}$ генерує *TV*-алгебру ATV_{3_1}, яка індукує алгебри *P*-предикатів.

Множина $TV_{3\_2} = \{T, F, TF\}$ генерує *TV*-алгебру ATV_{3_2}, яка індукує алгебри *T*-предикатів.

Множина $TV_{3\_3} = \{T, F, TF\uparrow\}$ генерує *TV*-алгебру ATV_{3_3}, яка індукує алгебри $TU_{=}A$-предикатів.

Множина $TV_{3\_D} = \{\uparrow, TF, TF\uparrow\}$ незамкнена щодо $\vee_*$: $TF \vee_* \uparrow = TF\uparrow \vee_* \uparrow = T\uparrow$.

Склад TV_{3_D} засвідчує *невідмінність* істини й фальші, що злилися воєдино!

Коректні модифікації $\vee_*$ на TV_{3_D}, для яких виконується *TFC*, *неможливі*.

Справді маємо:

$T(TF\vee\uparrow) = T(TF)\cup T(\uparrow) = {}^{V}A$;

$T(TF\uparrow\vee\uparrow) = = T(TF\uparrow)\cup T(\uparrow) = {}^{V}A$;

$F(TF\vee\uparrow) = F(TF)\cap F(\uparrow) = F(TF)\cap\varnothing = \varnothing$;
$F(TF\uparrow\vee\uparrow) = F(TF\uparrow)\cap F(\uparrow) = F(TF\uparrow)\cap\varnothing = \varnothing$.

Отже, *єдиною* модифікацією $\vee_{md}$ композиції $\vee_*$ за умови *TFC* є така:

$$TF \vee_{md} \uparrow = T \text{ та } TF\uparrow \vee_{md} \uparrow = T.$$

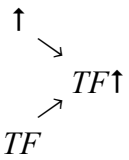
Проте $T \notin \{\uparrow, TF, TF\uparrow\}$.

Відносно "природними" модифікаціями $\vee_*$ для TV_{3_D} є $\vee_{eu}$ та $\vee_{\#}$, до певної міри $\vee_{\#D}$.

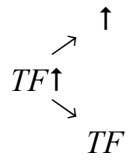
У випадку $\vee_{eu}$ маємо *TV*-алгебру ATV_{3_eu}, вона є підалгеброю *AEU*.

$\vee_{eu}$ та $\&_{eu}$ на TV_{3_D} збігаються.

Діаграма Хассе для TV_{3_D} щодо $\vee_{eu}$:



Діаграма Хассе для TV_{3_D} щодо $\&_{eu}$:



У випадку $\vee_{\#}$ маємо *TV*-алгебру $ATV_{3\_\#}$, вона є підалгеброю $AEU_{\#}$.

$\vee_{\#}$ та $\&_{\#}$ на TV_{3_D} збігаються.

Діаграма Хассе для TV_{3_D} щодо $\vee_{\#}$:

$$\uparrow \to TF \to TF\uparrow.$$

Діаграма Хассе для TV_{3_D} щодо $\&_{\#}$:

$$TF\uparrow \to TF \to \uparrow.$$

У випадку $\vee_{\#D}$ маємо *TV*-алгебру $ATV_{3\_\#D}$, вона є підалгеброю $AEU_{\#D}$.

$\vee_{\#D}$ та $\&_{\#D}$ на TV_{3_D} збігаються.

$\vee_{\#D}$ та $\&_{\#D}$ на TV_{3_D} неасоціативні, тому *TV*-алгебра $ATV_{3\_\#D}$ *цілком девіантна*.

Алгебри ATV_{3_eu}, $ATV_{3\_\#}$, $ATV_{3\_\#D}$ попарно неізоморфні, вони також неізоморфні жодній з 3-елементних алгебр ATV_{3_1}, ATV_{3_2}, ATV_{3_3}, ATV_{3_4}, ATV_{3_5}, $ATV_{TnU=A}$.

Через порушення умови *TFC* девіантні алгебри ATV_{3_eu}, $ATV_{3\_\#}$, $ATV_{3\_\#D}$ не індукують композиційні алгебри *GND*-предикатів із коректними $\vee$ та $\&$.

Маємо чотири **2-*елементних*** підмножини TV_{D5}, які замкнені щодо $\neg_*$.

Множина $TV_{2\_1} = \{T, F\}$ генерує класичну булеву *TV*-алгебру ATV_{2_1}, яка індукує алгебри *TS*-предикатів.

Множина $TV_{2\_3} = \{TF, TF\uparrow\}$ генерує *TV*-алгебру ATV_{2_3}, яка індукує алгебри *TAmG*-предикатів. Клас *TAmG*-предикатів вироджений [8].

Множина $TV_{2\_D} = \{\uparrow, TF\}$ незамкнена щодо $\vee_*$: $TF \vee_* \uparrow = T\uparrow$. Коректні модифікації $\vee_*$ із *TFC*, *неможливі*, адже *єдиною* модифікацією $\vee_{md}$ композиції $\vee_*$ за умови *TFC* є така: $TF \vee_{md} \uparrow = T$ та $TF\uparrow \vee_{md} \uparrow = T$.

Водночас $T \notin \{\uparrow, TF\}$.

Природні модифікації $\vee_*$ на TV_{2_D} – це $\vee_\#$ і дуальна їй на TV_{2_D} композиція $\vee_{\#d}$:
$\uparrow \vee_{\#d} \uparrow = TF \vee_{\#d} \uparrow = \uparrow \vee_{\#d} TF = TF$,
$TF\uparrow \vee_{\#du} TF\uparrow = TF\uparrow$.

У випадку $\vee_\#$ маємо *TV*-алгебру $ATV_{2\_D\#}$, вона ізоморфна алгебрі ATV_{2_3} (при перейменуванні $\uparrow$ на *TF* та *TF* на $TF\uparrow$).

У випадку $\vee_{\#d}$ маємо *TV*-алгебру ATV_{2_Dd}, вона ізоморфна ATV_{2_3} (при перейменуванні $\uparrow$ на $TF\uparrow$).

Множина $TV_{2\_U} = \{\uparrow, TF\uparrow\}$ незамкнена щодо $\vee_*$: $TF\uparrow \vee_* \uparrow = T\uparrow$. Коректні модифікації $\vee_*$, для яких виконується *TFC*, *неможливі*, адже *єдиною* модифікацією $\vee_{md}$ композиції $\vee_*$ за умови *TFC* є така:

$$TF \vee_{md} \uparrow = T \text{ та } TF\uparrow \vee_{md} \uparrow = T.$$

Проте $T \notin \{\uparrow, TF\uparrow\}$.

Природні модифікації $\vee_*$ на TV_{2_U} – це $\vee_\#$ і дуальна їй на TV_{2_U} композиція $\vee_{\#du}$:
$\uparrow \vee_{\#du} \uparrow = TF\uparrow \vee_{\#du} \uparrow = \uparrow \vee_{\#du} TF\uparrow = TF\uparrow$,
$TF\uparrow \vee_{\#du} TF\uparrow = TF\uparrow$.

У випадку $\vee_\#$ маємо *TV*-алгебру $ATV_{2\_U\#}$, вона ізоморфна ATV_{2_3} (при перейменуванні $\uparrow$ на *TF*).

У випадку $\vee_{\#du}$ маємо *TV*-алгебру ATV_{2_Ud}, вона ізоморфна алгебрі ATV_{2_3} (при перейменуванні $\uparrow$ на $TF\uparrow$ та $TF\uparrow$ на $\uparrow$).

Через порушення *TFC* девіантні алгебри $ATV_{2\_D\#}$, ATV_{2_Dd}, $ATV_{2\_U\#}$, ATV_{2_Ud} не індукують композиційні алгебри *GND*-предикатів із коректними $\vee$ та $\&$.

Висновки

Вивчаються програмно-орієнтовані логічні формалізми – логіки загальних недетермінованих (*GND*) предикатів. *GND*-предикати моделюються як 7-значні *TD*7-предикати. Множина TV_7 їх істиннісних значень задає алгебру істиннісних значень (*TV*-алгебру) $ATV_7 = (TV_7, \{\neg_*, \vee_*\})$. Багато підмножин TV_7 незамкнені щодо $\neg_*$ чи $\vee_*$, тому не утворюють підалгебр ATV_7. Такі підмножини та відповідні класи *GND*-предикатів названо девіантними. Девіантні *TV*-алгебри та девіантні класи *GND*-предикатів досліджено в даній роботі. Для того, щоб девіантна $TV \subseteq TV_7$ утворила алгебру, необхідно модифікувати $\neg_*$ чи $\vee_*$. В роботі не розглядається модифікація $\neg_*$, яка веде до специфічних некласичних логік. Найважливішими є такі модифікації $\vee_*$, для яких виконується умова *TFC* коректності логічних зв'язок предикатних алгебр. При порушенні умова *TFC* *TV*-алгебра стає девіантною, вона не індукує алгебру *GND*-предикатів. Для всіх підмножин TV_7 досліджено можливість модифікації $\vee_*$ із умовою *TFC*, що визначає відповідні класи *GND*-предикатів. Описано природні модифікації $\vee_*$ без умови *TFC*, що дає низку девіантних *TV*-алгебр. Для трьох девіантних множин не існує модифікацій $\vee_*$ із *TFC*, для них вказано відносно природні девіантні *TV*-алгебри.

Література

1. Handbook of Logic in Computer Science. Edited by S. Abramsky, Dov M. Gabbay and T. S. E. Maibaum. Oxford University Press, Vol. 1–5. 1993 – 2000.
2. Avron A., Zamansky A. Non-deterministic semantics for logical systems, in Handbook of Philosophical Logic, D.M. Gabbay, F. Guenthner (eds.), 2nd ed. 2011. Springer Netherlands. Vol. 16. P. 227–304.
3. Hähnle R. Many-valued logic, partiality, and abstraction in formal specification languages. *Logic Journal of the IGPL,* 2005. **13.** P. 415–433.
4. Jones C. Reasoning about partial functions in the formal development of programs. In: Proceedings of AVoCS'05. Elsevier,

Electronic Notes in Theoretical Computer Science. 2006. Vol. 145. P. 3–25.
5. Нікітченко М.С., Шкільняк С.С. Прикладна логіка. К.: ВПЦ Київський університет, 2013. 278 с.
6. Нікітченко М.С., Шкільняк О.С.. Шкільняк С.С. Чисті першопорядкові логіки квазіарних предикатів. *Проблеми програмування.* 2016. № 2–3. С. 73–86.
7. Mykola S. Nikitchenko and Stepan S. Shkilniak. Algebras and logics of partial quasiary predicates. Algebra and Discrete Mathematics. 2017. Vol. 23. N 2. P. 263–278.
8. Нікітченко М.С., Шкільняк О.С., Шкільняк С.С. Алгебри загальних недетермінованих предикатів. *Проблеми програмування.* 2018. № 1. С. 5–21.
9. Нікітченко М.С., Шкільняк О.С.. Шкільняк С.С. Логіки загальних недетермінованих предикатів: семантичні аспекти. *Проблеми програмування.* 2018. № 2–3. С. 31–45.
10. Belnap N., Steel T. The Logic of Questions and Answers. New Haven and London: Yale Univ. Press, 1976.
11. Нікітченко М.С., Шишацька О.В. Семантичні властивості п'ятизначних логік. *Проблеми програмування.* 2018. № 1. С. 22–35.

References

1. Abramsky S., Gabbay D. and Maibaum T. (editors). (1993–2000). Handbook of Logic in Computer Science Oxford University Press, Vol. 1–5.
2. Avron A. and Zamansky A. (2011). Non-deterministic semantics for logical systems. In Handbook of Philosophical Logic, D.M. Gabbay, F. Guenthner (eds.), 2nd ed., vol. 16, Springer Netherlands. P. 227–304.
3. Hähnle R. (2005). Many-valued logic, partiality, and abstraction in formal specification languages. *In Logic Journal of the IGPL*, **13**. P. 415–433
4. Jones C. (2006). Reasoning about partial functions in the formal development of programs. In Proceedings of AVoCS'05. V. 145. Elsevier, Electronic Notes in Theoretical Computer Science. P. 3–25.
5. Nikitchenko M. and Shkilniak S. (2013). Applied logic. Kyiv: VPC Kyivskyi Universytet (in ukr).
6. Nikitchenko M., Shkilniak O. and Shkilniak S. (2016). Pure first-order logics of quasiary predicates. *In Problems in Progamming*. N 2–3. P. 73–86 (in ukr).
7. Nikitchenko M. and Shkilniak S. (2017). Algebras and logics of partial quasiary predicates. In Algebra and Discrete Mathematics. Vol. 23. N 2. P. 263–278.
8. Nikitchenko M., Shkilniak O. and Shkilniak S. (2018). Algebras of general non-deterministic predicates. *In Problems in Progamming*. N 1. P. 5–21 (in ukr).
9. Nikitchenko M., Shkilniak O. and Shkilniak S. (2018). Logics of general non-deterministic predicates: semantic aspects. *In Problems in Progamming*. N 2–3. P. 31–45 (in ukr).
10. Belnap N. and Steel T. (1976). The Logic of Questions and Answers. New Haven and London: Yale Univ. Press.
11. Nikitchenko M., and Shyshatska E. (2018). Algebras of general non-deterministic predicates. *In Problems in Progamming*. N 1. P. 22–35 (in ukr).

Одержано 08.02.2019

Про авторів:

Шкільняк Оксана Степанівна,
кандидат фізико-математичних наук,
доцент, доцент кафедри
інформаційних систем.
Кількість наукових публікацій в
українських виданнях – понад 90,
у тому числі у фахових виданнях – 35.
Кількість наукових публікацій в
зарубіжних виданнях – 12.
Scopus Author ID: 57190873266
h-індекс (Google Scholar): 5 (4 з 2014)
http://orcid.org/0000-0003-4139-2525.

***Місце роботи автора*:**

Київський національний університет
імені Тараса Шевченка,
01601, Київ, вул. Володимирська, 60.
Тел.: (044) 259 05 19.
E-mail: me.oksana@gmail.com

УДК 004.4'24 https://doi.org/10.15407/pp2019.01.027

А.Ю. Дорошенко, М.М. Бондаренко, О.А. Яценко

АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ OPENCL ПРОГРАМ НА ОСНОВІ АЛГЕБРО-АЛГОРИТМІЧНОГО ПІДХОДУ

Подальший прогрес у покращенні показників якості створення паралельних програм пов'язаний з використанням гетерогенних архітектур обчислювальних систем. Неоднорідні паралельні системи включають у себе, зокрема, гібридні обчислювальні платформи, що поєднують використання центральних процесорів і графічних прискорювачів. Одним з інструментальних засобів програмування таких систем є OpenCL. У статті виконане налаштування раніше створеного алгебро-алгоритмічного інструментарію проектування і синтезу на автоматизовану разробку OpenCL-програм. Особливістю запропонованого підходу до проектування є використання мови, що ґрунтується на системах алгоритмічних алгебр Глушкова. Підхід продемонстровано на проектуванні програми інтерполяції для задачі метеорологічного прогнозування. Проведено експеримент з виконання згенерованої за допомогою розробленого інструментарію паралельної програми на графічному прискорювачі. Виконане порівняння з реалізацією програми на CUDA.

Ключові слова: автоматизоване проектування програм, алгебра алгоритмів, метеорологічне прогнозування, неоднорідні паралельні обчислювальні системи, синтез програм, CUDA, OpenCL.

Вступ

Паралельні та розподілені обчислення на мультипроцесорних платформах на даний час є основним джерелом забезпечення необхідних потреб у високій продуктивності обчислень при розв’язанні складних науково-технічних і господарських проблем. Подальший прогрес у покращенні показників якості створення паралельних програм пов’язаний із використанням неоднорідних архітектур обчислювальних систем. Неоднорідні паралельні системи включають у себе, зокрема, гібридні обчислювальні платформи, що поєднують використання центральних процесорів (Central Processing Units, CPUs) та графічних прискорювачів (Graphics Processing Units, GPUs). Одним з інструментальних засобів програмування таких систем є OpenCL (Open Computing Language) [1] – фреймворк для створення програмних застосувань, що виконуються у гетерогенному середовищі. На відміну від Nvidia CUDA [2], яка є реалізацією GPGPU (обчислення загального призначення на графічних процесорах) лише для відеокарт Nvidia, OpenCL є специфікацією, яку можуть реалізовувати різні виробники апаратного забезпечення. Існують реалізації OpenCL для відеокарт Nvidia, AMD, Intel, ARM, а також програмованих користувачем вентильних матриць (FPGA). Також на відміну від платформи Nvidia CUDA, програми для якої потрібно компілювати за допомогою спеціального компілятора, програми OpenCL можна збирати будь-яким компілятором, треба лише вказати бінарний файл бібліотеки при збірці. Зазначимо, що програмування гібридних систем – це досить складна задача порівняно з розробкою застосувань для однорідних архітектур, тому актуальним є питання розробки спеціальних засобів автоматизації розробки програмного забезпечення, що дозволяли б найбільш ефективно генерувати найпродуктивніший код для таких систем.

В роботах [3, 4] запропоновані теорія, методологія та інструментарій для автоматизованого проектування, синтезу та перетворення послідовних та паралельних програм, що ґрунтуються на засобах алгебр алгоритмів та переписувальних правил. В роботах [5–7] розроблені засоби застосовані для генерації паралельних програм для платформи Nvidia CUDA на основі схем алгоритмів. У даній роботі виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез програм, що використовують OpenCL. Застосування інструментарію продемонстроване на розробці паралельної програми інтерполяції, що

входить до складу програми чисельного прогнозування погоди [6, 7]. Проведено експеримент з виконання згенерованої паралельної OpenCL програми на графічному прискорювачі.

Запропонований у даній статті підхід є близьким до робіт, присвячених генерації OpenCL програм [8–11]. Зокрема, в роботі [8] розглядається підхід та програмний засіб Gaspard2 для специфікації, проектування та генерації вищезгаданих програм на основі використання уніфікованої мови моделювання UML. В роботі [9] запропоновані програмний засіб STEPOCL та предметно-орієнтована мова для спрощення розробки програм, що використовують декілька прискорювачів одночасно. В [10] розглядається автоматичний генератор функцій-ядер OpenCL для задач лінійної алгебри на основі високорівневих специфікацій, поданих користувачем у предметно-орієнтованій мові, що ґрунтується на C++. В ргототі [11] запропонований підхід до генерації OpenCL на основі високорівневих функціональних специфікацій та використання переписувальних правил.

Відмінність підходу, викладеного у даній статті, полягає у використанні для автоматизованого проектування паралельних програм специфікацій систем алгоритмічних алгебр Глушкова, поданих у природно-лінгвістичній формі, що полегшує розуміння алгоритмів і досягнення необхідної якості програм. Іншою перевагою розроблених інструментальних засобів є застосування методу діалогового конструювання синтаксично правильних програм, що виключає можливість виникнення синтаксичних помилок у процесі проектування схем.

1. Алгебро-алгоритмічні засоби проектування та їх налаштування на розробку OpenCL програм

В основу пропонованого підходу до проектування паралельних програм покладений апарат систем алгоритмічних алгебр (САА) та їх модифікацій [3, 4]. Модифіковані САА (САА-М) призначені для формалізації процесів мультиобробки, що виникають при конструюванні програмного забезпечення в мультипроцесорних системах. На САА-М ґрунтуються розроблені інструментальні засоби автоматизованого проектування та генерації програм [3–7].

1.1. Системи алгоритмічних алгебр. Модифіковані САА є двоосновною алгеброю $< Pr, Op; \Omega >$, де Pr – множина логічних умов (предикатів); Op – множина операторів; Ω – сигнатура, що складається з логічних операцій (диз'юнкції, кон'юнкції, заперечення, лівого множення оператора на умову) та операторних операцій (композиції, альтернативи, циклу та ін.), що будуть розглянуті далі. Предикати та оператори можуть бути базисними або складеними. Базисні елементи вважаються атомарними, неподільними абстракціями та пов'язані з предметною областю алгоритму, що проектується. Складені оператори будуються з елементарних за допомогою операцій послідовного й паралельного виконання операторів.

На САА-М ґрунтується алгоритмічна мова САА/1 [5, 6], призначена для багаторівневого структурного проектування і документування послідовних та паралельних алгоритмів і програм. Перевагою її використання є можливість опису алгоритмів у природно-лінгвістичній формі. Подання операторів мовою САА/1 називаються САА-схемами.

Далі наведено перелік назв та специфікації основних операторних операцій сигнатури САА-М, поданих у природньо-лінгвістичній формі.

1. Композиція (послідовне виконання операторів):

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

2. Альтернатива (умовний оператор):

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. Синхронізатор, що виконує затримку обчислень доти, поки значення умови не стане істинним:

WAIT ‘*condition*’.

В роботі [5] розглядаються додаткові операції, призначені для формалізованого проектування програм, що використовують платформу Nvidia CUDA. В роботі [6] згадані засоби використані для проектування програми чисельного прогнозування погоди.

1.2. Розширення САА-М операціями, орієнтованими на паралельні обчислення в гетерогенному середовищі. В даному підрозділі розглядаються нові операції алгебри алгоритмів, призначені для проектування OpenCL програм [1]. OpenCL є технологією створення програмних застосувань, що виконуються у гетерогенному середовищі. OpenCL поєднує прикладний програмний інтерфейс (API) та варіант мови С для програмування та одночасного використання різних пристроїв, здатних виконувати паралельні обчислення (наприклад, CPU, GPU та Xeon Phi). Для координації роботи пристроїв у гетерогенному середовищі присутній один “головний” пристрій (host), який взаємодіє з усіма іншими обчислювальними пристроями (device) за допомогою OpenCL API. OpenCL програма працює з так званими платформами (platform). Платформа є програмним пакетом, який надається відповідним розробником апаратних засобів (наприклад, AMD Accelerated Parallel Processing, Nvidia CUDA, Intel OpenCL). Платформа OpenCL складається з головного пристрою, з’єднаного з обчислювальними пристроями, що підтримують OpenCL. Кожний OpenCL-пристрій складається з обчислювальних блоків (compute unit), які далі розділяються на один або більше елементів-обробників (processing elements, PE). Програма в OpenCL складається з двох частин: головної, яка виконується на керуючому пристрої, та обчислювальної – однієї або кількох функцій-ядер, що виконуються на елементах-обробниках OpenCL-пристроїв. Головна частина програми визначає середовище (контекст), у якому виконуються ядра, і керує їх виконанням. Контекст включає платформу, обчислювальні пристрої та буфери пам’яті для них. Для обчислювального пристрою в рамках контексту створюється черга команд для виконання (command queue). Операції з обчислювальним пристроєм, такі, як читання/запис даних і запуск ядра, заносяться в цю чергу і послідовно виконуються. Коли ядро ставиться в чергу на виконання, визначається простір індексів. Копія ядра виконується для кожного індексу з цього простору і називається робочою одиницею (задачею, work-item). Робочі одиниці організовані в групи (work-groups) і виконуються паралельно на елементі-обробнику обчислювального пристрою.

Далі наведено послідовність дій для створення OpenCL програм та відповідні нові базисні оператори, додані в алгебру алгоритмів.

1. Отримання списку доступних платформ та запис їх у змінну *pl*:

“Get all available platforms (*pl*)”.

2. Отримання списку пристроїв з отриманої платформи *pl*:

“Get all devices (*dvs*) available on a platform (*pl*)”.

3. Створення контексту виконання для пристроїв:

“Create a context (*cnt*) for devices (*dvs*)”.

4. Створення черги виконання для пристрою:

"Create a command queue (*cmdqueue*)
for a context (*cnt*) and a device (*dv*)".

5. Компіляція файлу, що містить вихідний код функції-ядра OpenCL:

"Create a kernel (*krnl*) from
a source (*program*)",

де *krnl* – назва функції-ядра; *program* – шлях до файлу з вихідним кодом цієї функції.

Наведемо приклад САА-схеми для функції-ядра, що виконує додавання двох векторів *a* та *b* довжини *n*:

SCHEME VECTOR ADD ====

"vecAdd(a, b, c, n)"
==== "Declare a variable (id) of type (int)";
(id := "Get the global work-item
identifier for dimension (0)");
IF (id < n)
THEN
"(c[id]) := (a[id] + b[id])"
END IF;

END OF SCHEME VECTOR ADD

У схемі базисний оператор "Get the global work-item identifier for dimension (0)" повертає глобальний ідентифікатор задачі.

6. Створення буферу для даних, зазначених у змінній *var*:

"Create a memory buffer (*buff*)
for data (*var*) on devices in
the context (*cnt*)".

7. Завантаження буферу для змінної *var* в пам'ять пристрою за допомогою черги:

"Add commands to queue (*cmdqueue*)
writing a buffer of data (*var*) from
host to device".

8. Установка значення *arg_value* для параметра під номером *arg_index* ядра *krnl*:

"Set the argument value (*arg_value*)
for a parameter (*arg_index*) of
a kernel (*krnl*)".

9. Завантаження ядра *krnl* в чергу пристрою та його асинхронне виконання:

"Add a command to queue
(*cmdqueue*) executing a kernel
(*krnl*)(*globalworksize*)
(*localworksize*) on a device",

де *globalworksize* – загальна розмірність простору індексів; *localworksize* – розмірність локальної підмножини задач у групі.

10. Синхронізація (очікування завершення) виконання усіх задач, що були занесені в чергу:

WAIT 'All previously queued
commands in (*cmdqueue*)
are issued to the device and
have completed'.

11. Зчитування даних з буферу, в який записувався результат:

"Add commands to queue
(*cmdqueue*) reading from a buffer
of data (*var*) from device to host".

Приклад використання вищенаведених конструкцій розглядається у розділі 2.

1.3. 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) [3, 4]. 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. Основна ідея методу полягає у порівневому конструюванні схем зверху

вниз шляхом 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.

Основними компонентами системи ІПС є такі:

- діалоговий конструктор синтаксично правильних програм (ДСП-конструктор), призначений для діалогового проектування схем алгоритмів та синтезу програм мовами Java та C, C++;
- редактор граф-схем;
- база даних алгеброалгоритмічних специфікацій, у якій зберігається текст конструкцій САА-М і базисних елементів схем, а також їх програмні реалізації;
- генератор САА-схем за гіперсхемами.

Для автоматизації виконання трансформацій алгоритмів система ІПС може застосовуватися спільно з системою TermWare [12, 13], що ґрунтується на парадигмі переписувальних правил. В роботі [7] на основі використання TermWare розроблено програмний засіб для оптимізації обчислень, що дозволяє в напівавтоматичному режимі здійснювати паралелізацію циклічних операторів програм, що використовують Nvidia CUDA.

З метою налаштування системи ІПС на проектування і генерацію OpenCL програм, в базу даних системи включені нові конструкції, розглянуті вище в підрозділі 1.2.

2. Приклад застосування системи ІПС для проектування OpenCL програми

У даному розділі розглядається використання інтегрованого інструментарію для автоматизованої розробки паралельної OpenCL програми інтерполяції початкових метеорологічних даних, що входить до складу програмного забезпечення чисельного прогнозування погоди [6, 7].

Далі наведено фрагмент початкової послідовної САА-схеми інтерполяції, побудованої в [7].

```
SCHEME INTERPOLATION
SEQUENTIAL

“interpolation_sequential”
==== FOR (h FROM 0 TO Pk–1)
       LOOP
         FOR (k FROM 0 TO Lmz–1)
         LOOP
           FOR (j FROM 0 TO Mmz–1)
           LOOP
             FOR (i FROM 0 TO Nmz–1)
             LOOP
               “(a) := ((WZZ + US[h][k][j][i] /
                0.321f) * Rs * VS[h][k][j][i])”;
               “(Tp) := (TS[h][k][j][i] *
                pow(1000.f / HS[h][k][j][i],
                2.f/7.f))”;
               “(Tv) := (Tp * (1.f + 0.6078f *
                QS[h][k][j][i]))”;
               “(Qc[h][k][j][i]) := (a – (0.5f *
                Tv + (1.f – Zmz[k]) * g *
                F_X[j][i] / 0.321f))”
             END OF LOOP
           END OF LOOP
         END OF LOOP
       END OF LOOP;

END OF SCHEME
INTERPOLATION SEQUENTIAL
```

Наведена схема є чотирма вкладеними циклами за ітераторами $h \in [0 \ldots$ $\ldots Pk-1]$, $k \in [0 \ldots Lmz-1]$, $j \in [0 \ldots Mmz-$ $-1]$, $i \in [0 \ldots Nmz-1]$, де Pk, Lmz, Mmz, Nmz – натуральні числа, що визначають розмір скінченно-різницевої сітки. У схемі US, VS, TS, HS, QS – вхідні масиви метеорологічних величин (горизонтальні складові вектору швидкості вітру, температура, вологість і т. п.); Qc – вихідний масив. Оскільки ітерації у циклах є незалежними, вони можуть бути виконані паралельно.

Проектування паралельної OpenCL програми за допомогою системи ІПС включає у себе розробку САА-схем для

функції-ядра та керуючої програми, а також генерацію на їх основі відповідного коду мовою програмування.

Далі наведена САА-схема для функції-ядра, сконструйована за допомогою інструментарію ІПС на основі перетворення послідовної схеми. У параметрах, вказаних у дужках після назви функції (interpolation) зазначені вхідні та вихідні дані для задачі. Розпаралелювання здійснюється за індексами h, k, j (їх значення обчислюється за допомогою оператора "Get the global work-item identifier for dimension"). За індексом i в ядрі виконується цикл основних обчислень.

```
SCHEME INTERPOLATION KERNEL

"interpolation(US, VS, HS, QS, TS, F_x,
Zmz, Qc, Pk, Lmz, Mmz, Nmz)"
==== "Declare the list of variables (h, k, j) of
        type (int)";
      (h := "Get the global work-item
            identifier for dimension (0)");
      (k := "Get the global work-item
            identifier for dimension (1)");
      (j := "Get the global work-item
            identifier for dimension (2)");
      IF NOT((h >= Pk) OR (k >= Lmz) OR
           (j >= Mmz))
      THEN
        FOR (i FROM 0 TO Nmz–1)
        LOOP
          "Declare a constant (ind) of type
           (unsigned int)";
          "Declare the list of variables
           (a, Tp, Tv) of type (float)";
          "(ind) := (h + Pk * k + Pk * Lmz *
            j + Pk * Lmz * Mmz * i)";
          "(a) := ((WZZ + US[ind] / 0.321f)
            * Rs * VS[ind])";
          "(Tp) := (TS[ind] *
           pow(1000.f / HS[ind], 2.f / 7.f))";
          "(Tv) := (Tp * (1.f + 0.6078f *
                  QS[ind]))";
          "(Qc[ind]) := (a – (0.5f * Tv +
                       (1.f – Zmz[k]) * g *
                       F_x[j + i * Mmz] /
                                  0.321f))"
        END OF LOOP
      END IF;

END OF SCHEME INTERPOLATION
KERNEL
```

На відміну від реалізації алгоритму для CUDA [7], в якій дані US, VS, HS, QS, TS, Qc зберігалися як чотиривимірні вказівники, що є неефективним і не дозволяє використовувати кешування даних процесором, в реалізації алгоритму на OpenCL, що розглядається у даній роботі, дані передаються суцільним блоком (continuous array). Індекс для обробки поточного елемента масиву обчислюється спеціальним чином та заноситься у змінну ind. При розпаралелюванні послідовного алгоритму звертання до елементів згаданих масивів (наприклад, US[h][k][j][i]) замінюється на відповідний запис з індексом ind (наприклад, US[ind]).

Далі наведена САА-схема головної частини програми, побудована на основі кроків 1–11, розглянутих у підрозділі 1.2.

```
SCHEME INTERPOLATION HOST

"main(argc, argv)"
==== "Read input parameters
      (Pk, Lmz, Mmz, Nmz)
      from command line (argc, argv)";
      "Get all available platforms (pl)";
      "Get all devices (dvs) available on a
       platform (pl[0])";
      "Create a context (cnt) for
       devices (dvs)";
      "Create a command queue
       (cmdqueue) for a context (cnt) and a
       device (dvs[0])";
      "Create a kernel ("interpolation")
       from a source
       ("kernels/interpolation.cl")";
      "Declare and fill 4D continuous arrays
       (US, VS, HS, QS, TS, Qc)
       of type (auto) and size
       (Pk, Lmz, Mmz, Nmz)";
      "Declare and fill a 2D continuous
```

```
    array (F_X) of type (auto)
    and size (Mmz, Nmz)";
  "Declare and fill a 1D continuous
    array (Zmz) of type (auto) and
    size (Nmz)";
  "Create memory buffers for data
    (US, VS, HS, QS, TS, Qc, F_X,Zmz)
    on devices in the context (cnt)";
  "Add commands to queue (cmdqueue)
    writing buffers of data
    (US, VS, HS, QS, TS, F_X, Zmz)
    from host to device";
  "Set the argument values for
    parameters of a kernel (krnl)";
  "Add a command to queue
    (cmdqueue) executing a kernel
    (krnl)(3)({Pk, Lmz, Mmz})
    on a device";
  WAIT 'All previously queued
    commands in (cmdqueue) are issued
    to the device and have completed';
  "Add commands to queue (cmdqueue)
    reading from a buffer of data (Qc)
    from device to host";

END OF SCHEME INTERPOLATION
HOST
```

На основі побудованих САА-схем в системі ІПС виконана генерація програмного коду OpenCL, результати виконання якого на графічному прискорювачі наведено у наступному розділі.

3. Результати експерименту

Проведено експеримент з виконання розробленої паралельної програми в обчислювальному середовищі, що складається з двоядерного процесора Intel Core i5-4210U (частота 1,7–2,7 ГГц) та графічного процесора nVidia GeForce 840M (384 ядра, 2 Гб оперативної пам'яті). Для порівняння виконано також відповідну програму, що використовує Nvidia CUDA [7].

На рис. 1 показано графік залежності часу виконання програми інтерполяції від розміру вхідних даних на CPU, а також програм CUDA та OpenCL. На рис. 2 показано графіки залежності мультипроцесорного прискорення

$$Sp = \frac{T_{CPU}}{T_{GPU}},$$

де T_{CPU}, T_{GPU} — час виконання на CPU та GPU, відповідно.

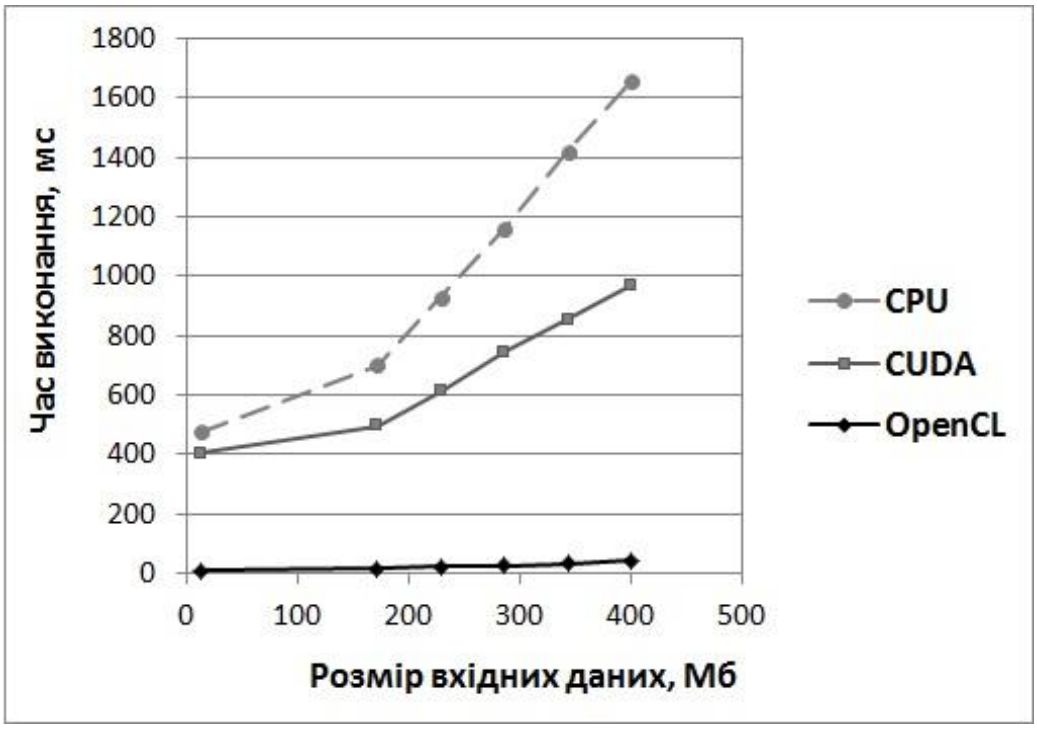


Рис. 1. Залежність часу виконання програми інтерполяції на CPU, CUDA та OpenCL від розміру вхідних даних

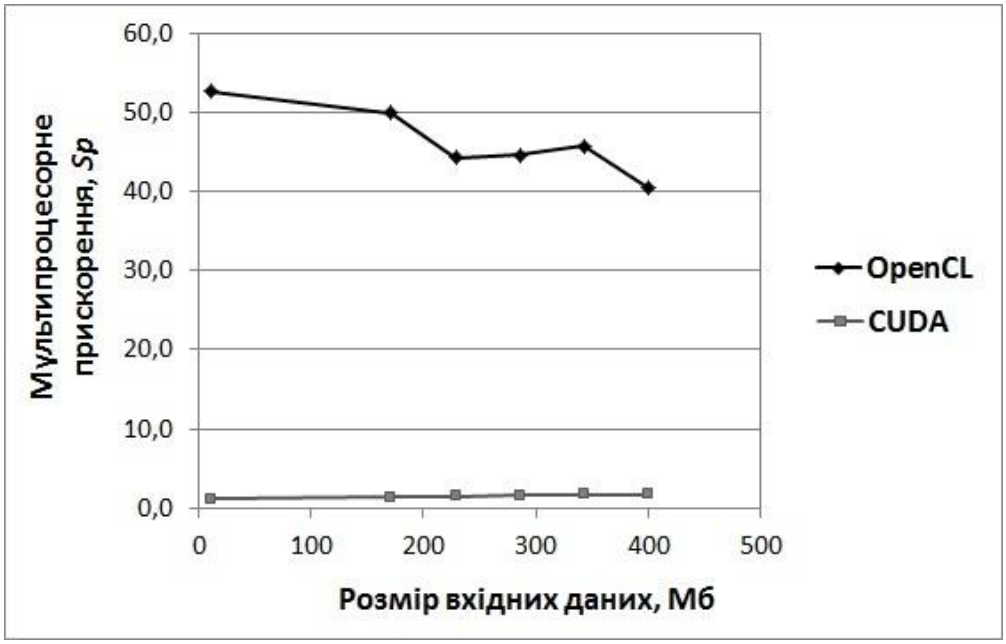


Рис. 2. Залежність мультипроцесорного прискорення від розміру вхідних даних для реалізацій паралельної програми інтерполяції на OpenCL та CUDA

Як видно з наведених результатів, OpenCL програма значно випереджає відповідний варіант для CUDA. Максимальне значення прискорення *Sp* OpenCL програми становить 52,7 (за розміру вхідних даних 11,451 Мб), для CUDA програми максимальне значення *Sp* є 1,7 (за розміру вхідних даних 400,553 Мб). Підвищена швидкодія OpenCL програми порівняно з реалізацією для CUDA пояснюється використанням одновимірних масивів даних замість чотиривимірних (див. розділ 2).

Висновки

Виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез OpenCL програм. Особливістю запропонованого підходу до проектування є використання мови САА-схем, перевагою якої є простота в навчанні і використанні, а також застосування методу конструювання синтаксично правильних схем, який виключає можливість появи синтаксичних помилок у процесі проектування алгоритмів.

Проведено експеримент з виконання згенерованої за допомогою системи ІПС паралельної програми інтерполяції початкових даних для задачі метеорологічного прогнозування на графічному прискорювачі. Результати експерименту продемонстрували гарний ступінь розпаралелюваності обчислень та переваги використання OpenCL порівняно з Nvidia CUDA для згаданої задачі.

Література

1. OpenCL overview. The open standard for parallel programming of heterogeneous systems [Електронний ресурс]. Режим доступу до ресурсу: https://www.khronos.org/opencl (дата звернення 22.01.2019 р.).
2. Nvidia CUDA technology [Електронний ресурс]. Режим доступу до ресурсу: http://www.nvidia.com/cuda (дата звернення 22.01.2019 р.).
3. 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.
4. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгеброалгоритмические модели и методы параллельного программирования. Киев: Академпериодика, 2007. 631 с.
5. Дорошенко А.Ю., Бекетов О.Г., Жереб К.А., Яценко О.А. Формалізоване проектування та синтез паралельних програм для

відеографічних прискорювачів. *Проблеми програмування.* 2013. № 3. С. 38–46.
6. Дорошенко А.Ю., Бекетов О.Г., Прусов В.А., Тирчак Ю.М., Яценко О.А. Формалізоване проектування та генерація паралельної програми чисельного моделювання погоди. *Проблеми програмування.* 2014. № 2–3. С. 72–81.
7. Дорошенко А.Ю., Яценко О.А., Бекетов О.Г. Алгоритм автоматизованого розпаралелювання циклічних операторів для графічних прискорювачів. *Проблеми програмування.* 2017. № 4. С. 28–36.
8. Rodrigues A., Guyomarc'h F., Dekeyser J.-L. An MDE approach for automatic code generation from UML/MARTE to OpenCL. *Computing in Science and Engineering*. 2012. Vol. 15, N 1. P. 46–55.
9. Li P., Brunet E., Trahay F., Parrot C., Thomas G., Namyst, R. Automatic OpenCL code generation for multi-device heterogeneous architectures. *Proc. 44th International Conference on Parallel Processing* (ICPP 2015), Beijing, China (1–4 September, 2015). Piscataway, NJ: IEEE, 2015. P. 959–968.
10. Tillet P., Rupp K., Selberherr S. An automatic OpenCL compute kernel generator for basic linear algebra operations. *Proc. 2012 Symposium on High Performance Computing* (HPC'12), Orlando, Florida, USA (26–30 March, 2012). San Diego, CA: Society for Computer Simulation International, 2012. P. 4:1–4:2.
11. Steuwer M., Fensch C., Lindley S., Dubach C. Generating performance portable code using rewrite rules: from high-level functional expressions to high-performance OpenCL code. *Proc. 20th ACM SIGPLAN International Conference on Functional Programming* (ICFP'15), Vancouver, Canada (31 August – 2 September, 2015). New York: ACM, 2015. P. 205–217.
12. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. *Fundamenta Informaticae*. 2006. Vol. 72, N 1–3. P. 95–108.
13. 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.

References

1. OpenCL Overview. The open standard for parallel programming of heterogeneous systems. [Online] Available from: https://www.khronos.org/opencl [Accessed: 22 January 2019]
2. Nvidia CUDA technology. [Online] Available from: http://www.nvidia.com/cuda [Accessed: 22 January 2019]
3. Andon, P.I. et al. (2017) Methods of algebraic programming. Formal methods of parallel program development. Kyiv: Naukova dumka. (in Russian)
4. Andon P.I. et al. (2007) Algebra-algorithmic models and methods of parallel programming. Kyiv: Akademperiodyka. (in Russian)
5. Doroshenko A.Yu., Beketov O.G., Zhereb K.A. & Yatsenko O.A. (2013) Formalized designing and synthesis of parallel programs for graphics processing units. Problems in programming. (3). P. 38–46. (in Ukrainian)
6. Doroshenko A.Yu., Beketov O.G., Prusov V.A., Tyrchak Yu.M. & Yatsenko O.A. (2014) Formalized designing and generation of parallel program for numerical weather forecasting task. *Problems in programming.* (2–3). P. 72–81. (in Ukrainian)
7. Doroshenko A.Yu., Yatsenko O.A. & Beketov O.G. (2017) Algorithm for automatic loop parallelization for graphics processing units. *Problems in programming*. (4). P. 28–36. (in Ukrainian)
8. Rodrigues A., Guyomarc'h F. & Dekeyser J.-L. (2012) An MDE approach for automatic code generation from UML/MARTE to OpenCL. Computing in Science and Engineering. 15 (1). P. 46–55.
9. Li P., Brunet E., Trahay F., Parrot C., Thomas G. & Namyst R. (2015) Automatic OpenCL code generation for multi-device heterogeneous architectures. In Proc. 44th International Conference on Parallel Processing (ICPP 2015). Beijing, China, 1–4 September 2015. Piscataway, NJ: IEEE. P. 959–968.
10. Tillet P., Rupp K. & Selberherr S. (2012) An automatic OpenCL compute kernel generator for basic linear algebra operations. In Proc. 2012 Symposium on High Performance Computing (HPC'12). Orlando, Florida, USA, 26-30 March 2012. San Diego, CA: Society for Computer Simulation International. P. 4:1–4:2.

11. Steuwer M., Fensch C., Lindley S. & Dubach C. (2015) Generating performance portable code using rewrite rules: from high-level functional expressions to high-performance OpenCL code. In Proc. 20th ACM SIGPLAN International Conference on Functional Programming (ICFP'15). Vancouver, Canada, 31 August – 2 September 2015. New York: ACM. P. 205–217.
12. Doroshenko A. & Shevchenko R. (2006) A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae. 72 (1–3). P. 95–108.
13. 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.

Одержано 10.02.2019

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу
теорії комп'ютерних обчислень,
професор кафедри автоматики та
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 Хірша – 5.
http://orcid.org/0000-0002-8435-1451,

Бондаренко Микола Миколайович,
магістр факультету комп'ютерних наук
та кібернетики.
http://orcid.org/0000-0002-6362-414X,

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в
українських виданнях – 43.
Кількість наукових публікацій в
зарубіжних виданнях – 13.
Індекс Хірша – 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.

Київський національний університет
імені Тараса Шевченка,
01601, Київ, вул. Володимирська, 60.

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

УДК 004.4'2 https://doi.org/10.15407/pp2019.01.037

О.М. Овдій

ДО ПИТАННЯ АВТОМАТИЗАЦІЇ ПРОЕКТУВАННЯ РОБОЧИХ ПРОЦЕСІВ НА ОСНОВІ АЛГЕБРО-АЛГОРИТМІЧНОГО ТА ОНТОЛОГІЧНОГО ІНСТРУМЕНТАРІЮ

Представлено концепцію системи для автоматизації проектування робочих процесів на основі алгебро-алгоритмічного та онтологічного інструментарію. Проведено аналіз існуючих підходів та запропоновано архітектуру системи. Роботу системи проілюстровано на прикладі розробки робочого процесу для аналізу великих обсягів даних на розподіленій платформі Apache Hadoop. Показано, що поєднання інструментів онтологій та алгебро-алгоритмічних інструментів забезпечує значний потенціал для адаптації, оптимізації, інтеграції та модифікації. Результати кількісної оцінки ефективності запропонованих засобів свідчать про те, що вони здатні сприяти істотному підвищенню продуктивності розробки і зменшенню трудовитрат.

Ключові слова: робочі процеси, онтологія, алгебра алгоритмів, проектування і синтез програм, розподілені обчислення.

Вступ

В останні роки активно розвиваються та розповсюджуються розподілені обчислювальні платформи. Зокрема це пов’язано з величезною кількістю цифрових даних, які потребують обробки і обсяг цих даних стрімко зростає з кожним днем. Поряд з цим, із зростом можливостей зростає і складність систем, а відповідно складність розробки програм для них. Сучасні програмні системи зазвичай є комплексними та складаються з різнорідних сервісів, що реалізовані на базі різних технологій, різними мовами програмування та розподілені у мережі. Крім того програми функціонують у гетерогенному середовищі, що швидко еволюціонує, інфраструктура змінюється, сервіси з’являються, змінюються та зникають. Одним з інструментів для спрощення проектування та підтримки таких програмних систем є робочі процеси (workflow) та системи керування ними. Формальна концепція робочого процесу існує вже протягом тривалого часу. Згідно з визначенням Workflow Management Coalition (WfMC) [1] робочий процес це автоматизація бізнес-процесу, цілком або частково, протягом якого документи, інформація або завдання передаються від одного учасника до іншого для дії, згідно з набором процедурних правил. У відношенні до розподілених обчислювальних платформ робочі процеси можна визначити як оркестрацію набору дій для досягнення більшої і складнішої мети [2]. Сьогодні системи керування робочими процесами широко використовуються як у бізнесі, так і у науковому секторі. Робочі процеси застосовуються в багатьох наукових дисциплінах, часто використовуючи великі за обсягом та різноманітні ресурси даних, а також паралельні та розподілені обчислювальні платформи. Робочі процеси забезпечують систематичний спосіб опису необхідних методів та інтерфейс між фахівцями та обчислювальним середовищем. Завдяки різкому збільшенню обсягів первинних даних робочі процеси відіграють все більш важливу роль, дозволяючи науковцям формулювати методи обробки та аналізу даних з різних джерел на широкому спектрі обчислювальних платформ. Наукова діяльності є дослідницькою за своїм характером і часто виконується методом "проб і помилок" а це потребує ще більшої адаптації. Крім того предметні експерти та науковці зазвичай не є професійними програмістами і їм потрібні інструменти, якими вони зможуть з легкістю користуватися. В зв’язку з цим виникає необхідність створення спеціальних високорівневих засобів проектування робочих процесів.

У даній роботі запропоновано концепцію системи для автоматизації проектування робочих процесів на основі алгебро-алгоритмічного та онтологічного інструментарію ОКРП. Дана система є подальшим розвитком методологій та інструментів, що ґрунтуються на засобах високорівневої алгебро-алгоритмічної формалізації і автоматизації перетворень програм [2–10]. Зокрема експериментальної інструментальної системи ІПС для конструювання та оптимізації паралельних програм, а також її онлайн версії системи ОДСП.

Основна відмінність інструментарію ОКРП у порівнянні з попередніми системами полягає в орієнтації на проектування робочих процесів та зберігання розроблюваних робочих процесів як екземплярів онтології у форматі OWL [11].

1. Підхід до проектування робочих процесів на основі онтологій і засобів алгебри алгоритмів

Одним з перспективних напрямів у розробці та дослідженні систем паралельних і розподілених обчислень нині є побудова програмних абстракцій у вигляді алгебро-алгоритмічних мов і моделей. У роботах [2–10] запропоновані теорія, методологія та інструментарій для автоматизованого проектування паралельних програм, що ґрунтуються на засобах високорівневої алгебро-алгоритмічної формалізації та автоматизації перетворень програм.

Наразі розроблено кілька алгоритмічних алгебр, найвідомішими з яких є алгебра Дейкстри, алгебра схем Янова та алгоритміка програм, досліджена у працях В.М. Глушкова та його учнів. Використання систем алгоритмічних алгебр (САА) В.М. Глушкова надає потужні можливості для розвитку архітектурно- і мовно-незалежних засобів програмування для розподілених обчислювальних систем і мереж.

Представлення алгоритмів алгебро-алгоритмічними засобами дозволяє автоматизовано їх перетворювати та оптимізувати, а крім того представляти алгоритми у природно-лінгвістичній формі, що є більш зрозумілою.

Онтологічні засоби, в свою чергу, представляють знання семантично, забезпечують необхідний рівень абстракції, сприяють їх поширенню та полегшують інтеграцію з іншими системами.

Онтологія [12], яка спочатку була введена в IT як засіб забезпечення семантики в Semantic Web, нині широко застосовується для забезпечення семантики та механізмів комунікації, та структурування знань у різноманітних галузях IT, бізнесу та багатьох інших сферах людської діяльності. Онтології нині застосовуються фактично на всіх етапах життєвого циклу програмного забезпечення [13]. Зокрема онтології застосовуються для вимог інженерії, представлення об’єктної моделі домену, документування коду, опису бізнес-правил, тестування та ін.

Онтології можуть:

– додати необхідний рівень абстракції, що спростить перетворення та адаптацію алгоритмів;

– посприяти структуруванню знань;

– забезпечити механізм комунікації та інтеграції.

Аналіз існуючих онтологічних підходів до інженерії програмного забезпечення, а зокрема робочих процесів виявив відсутність загальноприйнятих стандартів. В рамках роботи проаналізовано такі онтології як OWL-WS [14], SciFlow [15], WDO-It [16] та ін. В ході аналізу виявлено недоліки існуючих підходів:

– недостатній рівень абстракції;

– вузька спеціалізація на конкретному домені;

– спроба забезпечити високий рівень абстракції, поряд з наслідуванням спеціалізованих онтологій, що були розроблені для інших цілей.

У зв'язку з переліченими вище недоліками у даній роботі замість використання існуючих онтологій було вирішено розроблену раніше прикладну онтологію для проектування програм [10] розширити для проектування робочих процесів. З метою подальшого розвитку інструментарію та для збільшення переліку підтримуваних платформ онтологія побудована з належним рівнем абстракції. Як інструменталь-

ний засіб розробки онтології обрана система Protégé [17], та використано мову опису онтологій OWL (Web Ontology Language) [11].

На рис. 1 показано фрагмент ієрархії концептів розробленої онтології. Вузлами графу є концепти, а дугами відношення наслідування між ними.

Підкласи класу Data представляють різні структури даних. Клас Construction відображає типи конструкцій: атомарні (AtomicConstruction) та складні (CompoundConstruction). Поняття складної конструкції відповідає поняттю вкладеного робочого процесу. Клас Operation відображає типи операції САА: оператори та предикати. Операції можуть бути базисними (BasicOperation) або складними (CompoundOperation). Класи Construction

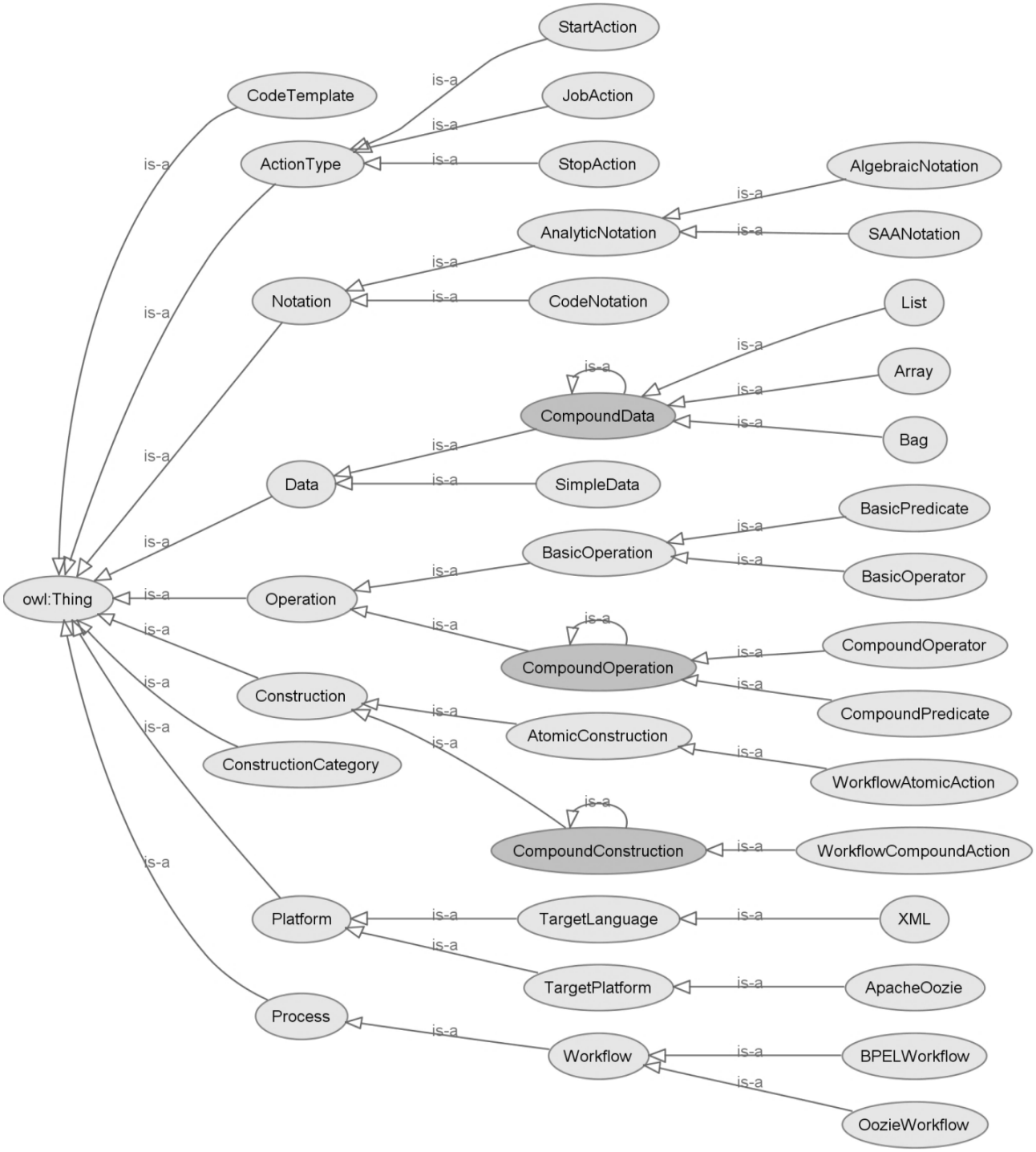


Рис. 1. Фрагмент ієрархії концептів онтології проектування робочих процесів

мають властивість isOperation, що ставить їм відповідний клас Operation для того щоб зв'язати конструкції робочого процесу з відповідними типами операцій в CAA. Клас Platform представляє підтримувані платформи та мови. ActionType – визначає тип дії робочого процесу, а саме може вона мати і вхідні і вихідні зв'язки, або тільки вхідні чи вихідні. Клас Notation представляє типи нотацій: алгебраїчну (AlgebraicNotation), природньо-лінгвістичну (SAANotation) та мовою програмування (CodeNotation). Клас Process призначений для специфікації алгоритмів, зокрема робочих процесів.

Включення семантики до специфікації робочого процесу дозволяє відокремлювати абстрактну специфікацію від фактичної реалізації, роблячи можливим переніс робочого процесу з однієї платформи керування робочими процесами на іншу та спрощує адаптацію до змін у середовищі виконання. Крім того, у зв'язку з специфікою наукових досліджень, наукові робочі процеси часто модифікуються та повторно використовуються.

З рухом у напрямку відкритої науки, виникає потреба зробити робочі процеси доступними не тільки їх авторам, а й іншим зацікавленим сторонам з різним рівнем підготовки від аматорів до професіоналів. Використання інструментів онтологій робити робочий процес більш зрозумілим для інших людей та програмних агентів.

В свою чергу, використання інструментів алгебри алгоритмів теж сприяє підвищенню рівня абстракції та підтримці гнучкості та платформо-незалежності. Окрім того дозволяє автоматизувати перетворення та оптимізацію робочих процесів.

Таким чином поєднання інструментів онтологій та алгебро-алгоритмічних інструментів забезпечує значний потенціал для адаптації, оптимізації, інтеграції та модифікації розроблюваних робочих процесів.

2. Онлайновий конструктор робочих процесів

В даній роботі представлено концепцію системи для автоматизації проектування робочих процесів на основі алгебро-алгоритмічного та онтологічного інструментарію: онлайновий конструктор робочих процесів (ОКРП).

Розробка даного інструменту була викликана необхідністю у подальшому розвитку інструментів автоматизованого проектування програм для підтримки проектування алгоритмів, що кодуються не звичайними мовами програмування, а визначаюся, наприклад, XML-специфікацією або спеціалізованими мовами програмування, зокрема робочих процесів.

В розроблюваній системі ОКРП, алгоритми робочих процесів представлені у графічній, онтологічній, аналітичній та природньо-лінгвістичній формах. Представлення алгоритму у аналітичній формі дозволяє виконувати перетворення, що може знадобитися для пристосування до змін у платформі або оптимізації. Графічна та природньо-лінгвістична форми роблять процес проектування алгоритмів зручним для людини, що полегшує досягнення необхідної якості програм, підвищує продуктивність та дозволяє уникати синтаксичних помилок. Онтологічне представлення та зберігання алгоритмів семантично їх збагачує, сприяє структуруванню знань та забезпечує механізм комунікації та інтеграції.

Основні особливості ОКРП:

– орієнтація на конструювання алгоритмів для робочих процесів та спеціалізованих мов програмування;

– графічний конструктор алгоритмів робочих процесів;

– форми представлення алгоритмів:

  - графічна,
  - аналітична,
  - природно-лінгвістична;

– зберігання розроблюваних алгоритмів як екземплярів онтології у форматі OWL.

Автоматизована система ОКРП є онлайновим сервісом, що призначений для проектування та генерації описів робочих процесів на основі високорівневих специфікацій алгоритмів. Система спрямована забезпечувати побудову робочих процесів у режимі порівневого конструювання та генерацію їх специфікації.

ОКРП ґрунтується на використанні розглянутих засобів CAA-M та онтологій.

Графічний інтерфейс діалогового конструктора орієнтований на зручність та легкість конструювання та виключення можливості появи синтаксичних помилок у процесі побудови алгоритму. Система базується на сервісно-орієнтованій архітектурі та спрямована на багатокористувальницьке використання інструментарію через мережу Інтернет. Розробка ведеться мовою Java.

Розроблювана автоматизована система ОКРП складається з компонентів, показаних на рис. 2.

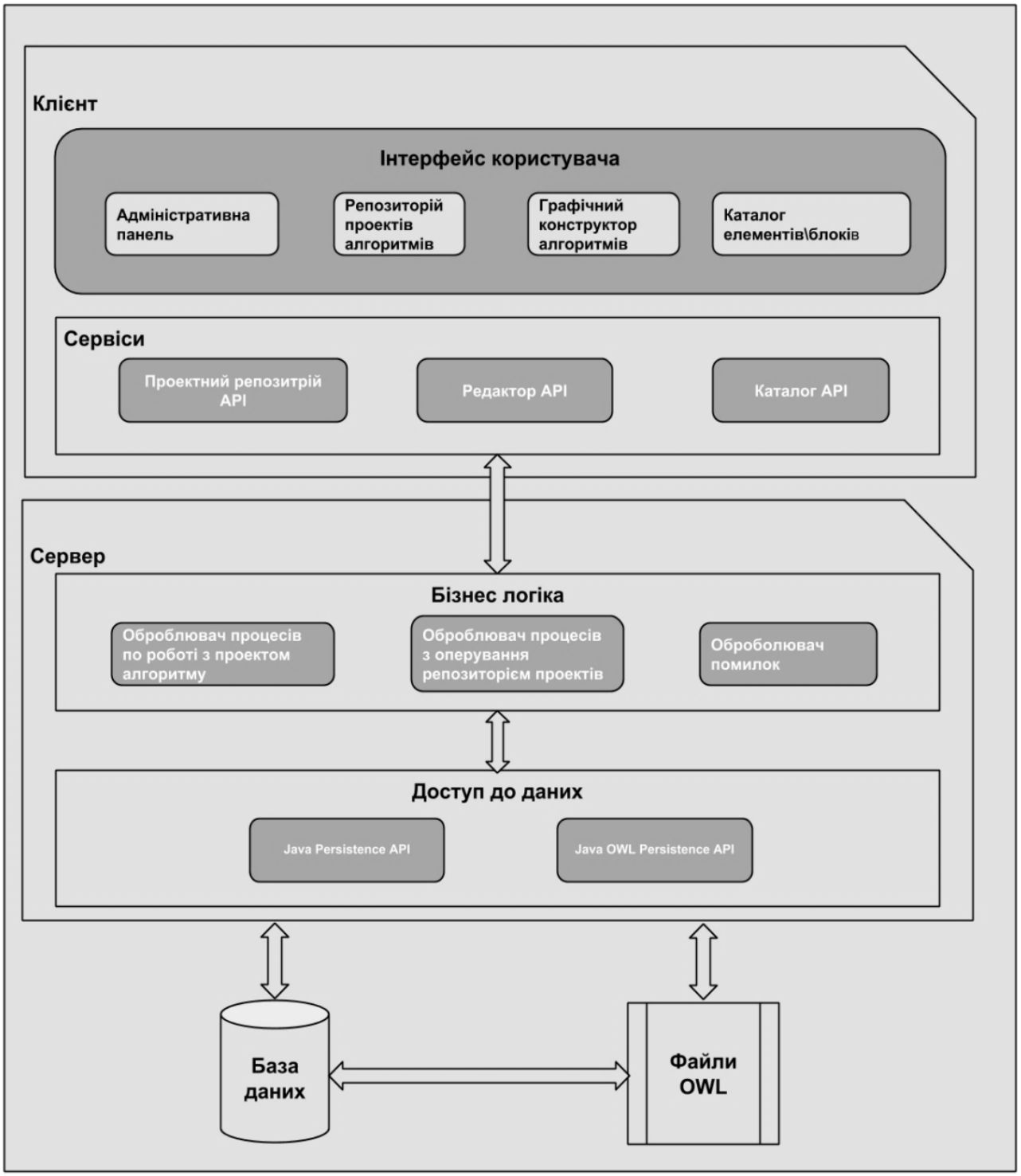


Рис. 2. Архітектура системи проектування робочих процесів

3. Приклад використання системи ОКРП

Задача обробки та аналізу великих обсягів даних надзвичайно актуальна у сфері метеорології. В ній наявні величезні обсяги історичних та поточних даних, а моделі та засоби аналізу надзвичайно складні та ресурсоємні. Наприклад, Національне управління океанічних і атмосферних досліджень США (NOAA) [18] щодня видає близько 20 терабайт даних, включаючи дані сотень метеорологічних станцій, результати прогнозів, дані супутників тощо. Для роботи з такими великими наборами даних науковцям необхідні інструменти, якими вони можуть користуватися без поглиблених знань програмування та архітектури обчислювальних систем.

Розглянемо функціонування системи ОКРП на прикладі розробки робочого процесу для аналізу великих обсягів даних на розподіленій платформі Apache Hadoop [19]. Apache Hadoop призначена для розподіленого зберігання й обробки великих обсягів даних на великих комп'ютерних кластерах, реалізація парадигми MapReduce. Робочий процес проектується для системи Apache Oozie [20]. Apache Oozie є системою керування робочими процесами для керування роботами Apache Hadoop. Для попередньої обробки даних використовується Apache Pig [21], що призначений для обробки великих наборів даних на Apache Hadoop. Для аналізу даних застосовується програмне середовище R [22] для статистичних обчислень та аналізу даних.

На рис. 3 показано інтерфейс розроблюваної системи з прикладом спроектованого алгоритму для робочого процесу Apache Oozie статистичного аналізу часових рядів у метеорологічних даних.

Робочий процес складається з наступних кроків:

– **Start** – запуск робочого процесу;

– **Load ncdc data** – вузол робочого процесу, який виконує запуск shell-скрипту, що завантажує сирі історичні данні метеорологічних станцій з сайту національного центру кліматичних даних США NCDC [18];

– **Prepare data** – запускає програму Apache Pig, що виконує попередню обробку даних. Виконується фільтрація даних за розташуванням метеостанції (м. Київ) та обчислюється середньомісячна температура за останні декілька років. Детальний приклад обробки даного масиву метеорологічних даних за допомогою Apache Pig розглянуто у попередній роботі [23].

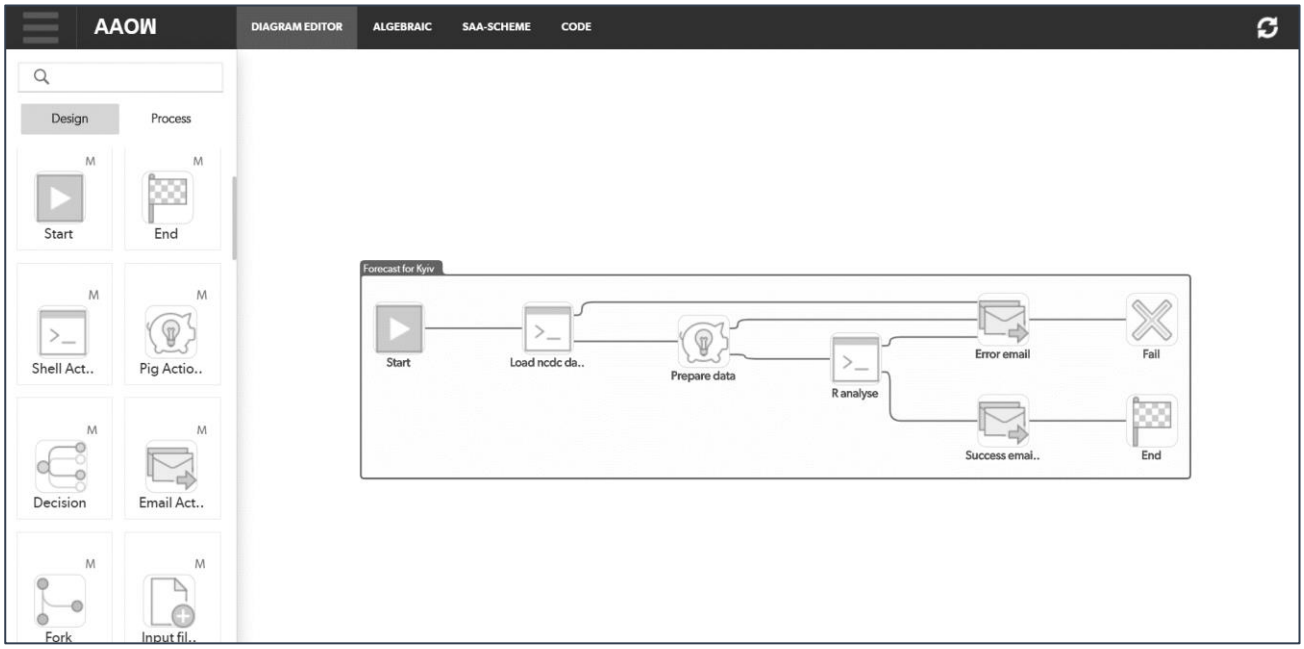


Рис. 3. Фрагмент копії екрану системи ОКРП

– **R analyse** – shell-скрипт який запускає на виконання програму R для статистичного аналізу даних та виводу результатів. Програма обчислює прогноз середньомісячної температури в м. Києві на наступні два роки за моделлю аналізу часових рядів ARIMA та графічно представляє результат.

– **Error email** – сповіщення у разі виникнення помилок та збою робочого процесу.

– **Success email** – сповіщення про вдале виконання робочого процесу та надсилання результатів.

Як кроки робочого процесу можна використовувати вже існуючі програми та сервіси або, при необхідності, розробити нові, зокрема за допомогою систем ІПС або ОДСП. Наприклад, програма Apache Pig, що використовується у кроці **Prepare data**, була розроблена у попередній роботі за допомогою системи ОДСП [23].

Далі наведено скорочений приклад алгебраїчного представлення спроектованого алгоритму для робочого процесу статистичного аналізу часових рядів у метеорологічних даних:

MID_TEMP_ANALYSE = Start THEN
Shell_action(upload-data, load_ncdc.sh, arguments, file) THEN
Pig_action(pig-prepare-data, prepare_data.pig, params, file) THEN
Shell_action(shell-r-analyse, run_r_hadoop.sh, arguments, file)
THEN End(end)

де **Start** – початок робочого процесу; **Shell_action(upload-data..** – вузол робочого процесу, який виконує запуск shell-скрипту, що завантажує сирі дані метеорологічних станцій; **Pig_action(pig-prepare-data..** – запускає програму Apache Pig, що виконує попередню фільтрацію та обробку даних; **Shell_action(shell-r-analyse..** - shell-скрипт, який запускає на виконання програму R для статистичного аналізу даних та виводу результатів.

Далі наведено приклад природно-лінгвістичного представлення попереднього алгоритму:

"MID_TEMP_ANALYSE"
==== "Start workflow"
THEN
"Run Shell Action upload-data with parameters: (load_ncdc.sh, arguments, file)"
THEN
"Run Pig Action pig-prepare-data with parameters (prepare_data.pig, params, file)"
THEN
"Run Shell Action shell-r-analyse with parameters: (run_r_hadoop.sh, arguments, file)"
THEN
"End of workflow (end)"

Далі системою буде згенеровано код специфікації робочого процесу для цільової системи керування робочими процесами, у даному випадку Apache Oozie (рис. 4).

Припустимо, що у постачальника метеорологічних даних змінився формат представлення даних. Для вирішення цієї проблеми в існуючий робочий процес після кроку **Load ncdc data** необхідно додати ще один крок, який буде виконувати конвертацію даних з нового формату. Інші кроки та їх взаємозв'язки не зміняться і для розробки та додавання нового кроку не потрібно вникати в деталі реалізації та функціонування всього робочого процесу, достатньо буде знати вимоги до формату даних. Крім того, використовуючи інструменти алгебри алгоритмів, можна буде автоматизовано перетворити всі існуючі робочі процеси та автоматично додати необхідний крок конвертації даних.

З наведеного прикладу видно, що робочі процеси для розподілених та гетерогенних середовищ з легкістю можна

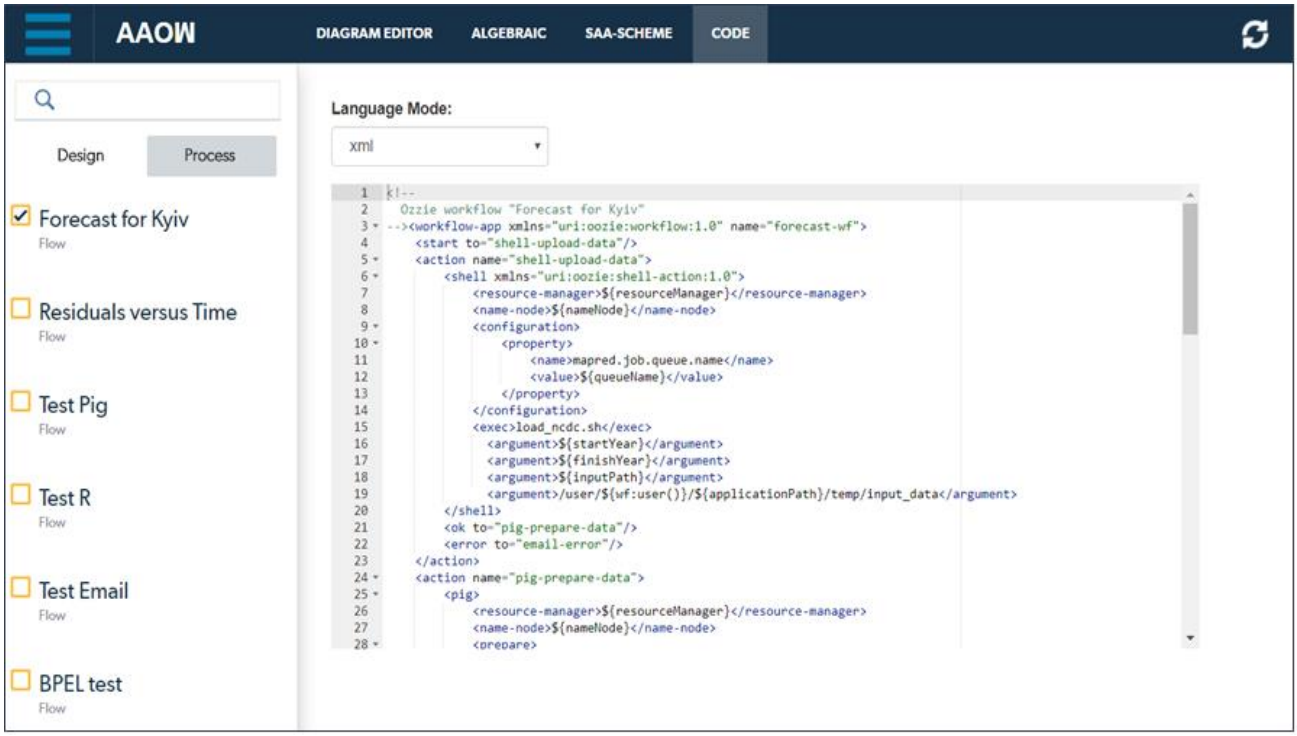


Рис. 4. Копія екрану системи ОКРП зі згенерованим кодом специфікації робочого процесу

проектувати та модифікувати за допомогою інструменту проектування робочих процесів ОКРП.

Зменшення обсягу коду необхідного для реалізації задачі зменшує трудовитрати на кодування і подальшу підтримку, та скорочує число можливих помилок. Длякількісної оцінки ефективності запропонованих засобів була використана метрика SLOC (Source Lines of Code, кількість рядків коду) [24], широко використовувана для оцінки трудовитрат на розробку програм. Для наведеного прикладу результати оцінки SLOC для САА-схеми і згенерованого за нею xml коду специфікації робочого процесу для цільової платформи Apache Oozie складає 16 і 88 відповідно. Таким чином, співвідношення рядків коду показує, що обсяг цільового коду перевищує обсяг САА-схеми більш ніж в 5 разів. Дані результати свідчать про те, що запропоновані засоби для автоматизованого проектування робочих процесів здатні сприяти підвищенню продуктивності розробки і зменшенню трудовитрат.

Висновки

В даній роботі представлено концепцію системи для автоматизації проектування робочих процесів на основі алгебро-алгоритмічного та онтологічного інструментарію: онлайновий конструктор робочих процесів ОКРП. Проведено аналіз існуючих підходів та запропоновано архітектуру системи. Роботу системи проілюстровано на прикладі розробки робочого процесу Apache Oozie для аналізу великих обсягів даних на розподіленій платформі Apache Hadoop. Результати кількісної оцінки ефективності запропонованих засобів свідчать про те, що вони здатні сприяти істотному підвищенню продуктивності розробки і зменшенню трудовитрат. Далі планується подальша розробка та розвиток системи ОКРП, зокрема розширення підтримкою інших стандартів та систем керування робочими процесами, зокрема BPEL.

Література

1. Workflow Management Coalition: сайт. URL: https://www.wfmc.org/ (дата звернення: 29.01.2019).

2. Amin K., Laszewski G., Hategan M., Zaluzec N. J., Hampton S., Rossi A. GridAnt: A Client-Controllable Grid Workflow System. In Proc. of the 37th Hawaii International Conference on System Sciences, HI, USA, 2004. P. 3293–3301.
3. Дорошенко А.Ю., Бекетов О.Г., Іванів Р.Б., Іовчев В.О., Мироненко І.О., Яценко О.А. Автоматизована генерація паралельних програм для графічних прискорювачів на основі схем алгоритмів. *Проблеми програмування.* 2015. № 1. С. 19–28.
4. Андон Ф.И., Дорошенко А.Е., Бекетов А.Г., Иовчев В.А., Яценко Е.А. Инструментальные средства автоматизации параллельного программирования на основе алгебры алгоритмов. *Кибернетика и системный анализ.* 2015. № 1. С. 162–170.
5. Дорошенко А.Ю., Іваненко П.А., Овдій О.М., Яценко О.А. Автоматизоване проектування програм для розв'язання задачі метеорологічного прогнозування. *Проблеми програмування.* 2016. № 1. С. 102–115.
6. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгеброалгоритмические модели и методы параллельного программирования. Киев: Академпериодика, 2007. 631 с.
7. Дорошенко Е.А., Яценко Е.А. О синтезе программ на языке Java по алгеброалгоритмическим спецификациям. *Проблеми програмування.* 2006. № 4. С. 58–70.
8. Яценко Е.А. Интеграция средств алгебры алгоритмов и переписывания термов для разработки эффективных параллельных программ. *Проблеми програмування*. 2013. № 2. С. 62–70.
9. Дорошенко А.Ю., Бекетов О.Г., Яценко О.А, Вітряк Є.А., Павлючин Т.О. Розробка сервісно-орієнтованих засобів для запуску паралельних програм на мультипроцесорному кластері. *Проблеми програмування*. 2014. № 4. С. 3–14.
10. Дорошенко Е.А., Овдей О.М., Яценко Е.А. Онтологические и алгеброалгоритмические средства автоматизации проектирования параллельных программ для "облачных" платформ. *Кибернетика и системный анализ.* 2017. Т. 53, № 2. С. 181–192.
11. OWL 2 Web Ontology Language Primer (Second Edition). URL: https://www.w3.org/2012/pdf/REC-owl2-primer-20121211.pdf (дата звернення: 29.01.2019).
12. Gruber T. R. A Translation Approach to Portable Ontologies. Knowledge Acquisition. 1993. N 5(2). P. 199–220.
13. Strmecki D., Magdalenic I., Kermek D. An Overview on the use of Ontologies in Software Engineering. *Journal of Computer Science* 2016. N 12(12). P. 597–610.
14. Beco S., Cantalupo B., Giammarino L., Matskanis N., Surridge M. OWL-WS: A Workflow Ontology for Dynamic Grid Service Composition. In: 1st Int. Conf. on e-Science and Grid Computing. IEEE Computer Society. 2005. P. 148–155.
15. Oliveira D., Ogasawara E., Araujo Baiao F., Mattoso M. Adding Ontologies to Scientific Workflow Composition. XXVI Simpósio Brasileiro de Banco de Dados. 2011. Florianópolis, SC. P. 147–154.
16. Pinheiro da Silva P., Salayandia L., Gates A. Q. WDO-It! A Tool for Building Scientific Workflows from Ontologies. Technical Report UTEP-CS-07-XX, University of Texas. 2007. URL: http://digitalcommons.utep.edu/ cs_techrep/201 (дата звернення: 29.01.2019).
17. Horridge M. A practical guide to building OWL ontologies using Protégé 4 and CO-ODE tools. Manchester: The University Of Manchester. 2011. 107 p.
18. National Climatic Data Center (NCDC): сайт. URL: https://www.ncdc.noaa.gov/ (дата звернення: 26.02.2018).
19. Apache Hadoop: сайт. URL: http://hadoop.apache.org/ (дата звернення: 29.01.2019).
20. Apache Oozie Workflow Scheduler for Hadoop: сайт. URL: http://oozie.apache.org/ (дата звернення: 29.01.2019).
21. Apache Pig: сайт. URL: http://pig.apache.org/ (дата звернення: 29.01.2019).
22. The R Project for Statistical Computing: сайт. URL: https://www.r-project.org/ (дата звернення: 29.01.2019).
23. Овдій О.М. Розширення системи синтезу програм з метою аналізу великих наборів даних. *Проблеми програмування* 2018. № 2-3. Спец. Вип. С. 68–74.
24. Nguyen V., Deeds-Rubin S., Tan T., Boehm B. A. SLOC Counting Standard. URL: http://csse.usc.edu/TECHRPTS/2007/usc-csse-2007-737/usc-csse-2007-737.pdf. (дата звернення: 29.01.2019).

References

1. Wfmc.org. Workflow Management Coalition. [online] Available from: https://www.wfmc.org/ [Accessed 29 Jan. 2019].
2. Amin, K., Laszewski, G., Hategan, M., Zaluzec, N. J., Hampton, S. & Rossi, A. (2004). GridAnt: A Client-Controllable Grid Workflow System. In: *37th Hawaii International Conference on System Sciences*. HI, USA. P. 3293–3301.
3. Doroshenko A.Yu., Beketov O.G., Ivaniv R.B., Iovchev V.O., Myronenko I.O. & Yatsenko O.A. (2015) Automated generation of parallel programs for graphics processing units based on algorithm schemes. *Problems in programming*. (1). P. 19–28. (in Ukrainian).
4. Andon P.I., Doroshenko A.Yu., Beketov O.G., Iovchev V.O. & Yatsenko O.A. (2015) Software tools for automation of parallel programming on the basis of algebra of algorithms. *Cybernetics and systems analysis*. (1). P. 162–170. (in Russian).
5. Doroshenko A.Yu., Ivanenko P.A., Ovdii O.M., & Yatsenko O.A. (2016) Automated design of programs for solving the task of meteorological forecasting. *Problems in programming*. (1). P. 102–115. (in Ukrainian).
6. Andon P.I. et al. (2007) *Algebra-algorithmic models and methods of parallel programming*. Kiev: Academperiodika. (in Russian).
7. Doroshenko A.Yu. & Yatsenko O.A. (2006) About the synthesis of Java programs by algebra-algorithmic specifications. *Problems in programming*. (4). P. 58–70. (in Russian).
8. Yatsenko O.A. (2013) Integration of algebra-algorithmic tools and term rewriting for efficient parallel programs development. *Problems in programming*. (2). P. 62–70. (in Russian).
9. Doroshenko A.Yu., Beketov O.G. Yatsenko O.A., Pavliuchyn T.O. & Vitriak I.A. (2014) Development of the service-oriented software for launching parallel programs on a multiprocessor cluster. *Problems in programming.* (4). P. 3–14. (in Ukrainian).
10. Doroshenko A.Yu., Ovdii O.M. & Yatsenko O.A. (2017) Ontological and algebra-algorithmic tools for automated design of parallel programs for cloud platforms. *Cybernetics and Systems Analysis*. 53(2). P. 181–192. (in Russian).
11. OWL 2 Web Ontology Language Primer (Second Edition). [online] Available from: https://www.w3.org/2012/pdf/REC-owl2-primer-20121211.pdf [Accessed 29 Jan. 2019].
12. Gruber T.R. (1993) A Translation Approach to Portable Ontologies. *Knowledge Acquisition*. 5(2). P. 199–220.
13. Strmecki D., Magdalenic I. & Kermek D. (2016) An Overview on the use of Ontologies in Software Engineering. *Journal of Computer Science*. 12(12). P. 597–610.
14. Beco S., Cantalupo B., Giammarino L., Matskanis N. & Surridge M. (2005) OWL-WS: A Workflow Ontology for Dynamic Grid Service Composition. In: *1st Int. Conf. on e-Science and Grid Computing*. IEEE Computer Society. P. 148–155.
15. Oliveira D., Ogasawara E., Araujo Baiao F. & Mattoso M. (2011) Adding Ontologies to Scientific Workflow Composition. In: *XXVI Simpósio Brasileiro de Banco de Dados.* Florianópolis, SC. P. 147–154.
16. Pinheiro da Silva, P., Salayandia L. & Gates A.Q. (2007) WDO-It! A Tool for Building Scientific Workflows from Ontologies. Technical Report UTEP-CS-07-XX, University of Texas. [online] Available from: http://digitalcommons.utep.edu/cs_techrep/201 [Accessed 29 Jan. 2019].
17. Horridge M. (2011) *A practical guide to building OWL ontologies using Protégé 4 and CO-ODE tools*. Manchester: The University Of Manchester. 2011.
18. Ncdc.gov. *National Climatic Data Center (*NCDC*).* [online] Available from: https://www.ncdc.noaa.gov/ [Accessed 29 Jan. 2019].
19. Hadoop.apache.org. *Apache Hadoop Official Website*. [online] Available from: http://hadoop.apache.org/ [Accessed 29 Jan. 2019].
20. Oozie.apache.org. *Apache Oozie Workflow Scheduler for Hadoop Official Website*. [online] Available from: http://oozie.apache.org/ [Accessed 29 Jan. 2019].
21. Pig.apache.org. *Apache Pig Official Website*. [online] Available from: http://pig.apache.org/ [Accessed 29 Jan. 2019].
22. R-project.org. *The R Project for Statistical Computing Official Website.* [online] Available from: https://www.r-project.org/ [Accessed 29 Jan. 2019].
23. Ovdii, O.M. (2018) Extension of the program synthesis system to analyze large data sets.

Problems in programming. (2-3). P. 68–74. (in Ukrainian).
24. Nguyen V., Deeds-Rubin S., Tan T., Boehm B.A. SLOC Counting Standard. [online] Available from: http://csse.usc.edu/TECHRPTS/2007/usc-csse-2007-737/usc-csse-2007-737.pdf. [Accessed 29 Jan. 2019].

Одержано 31.01.2019

***Про автора*:**

Овдій Ольга Михайлівна,
молодший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 22.
Кількість наукових публікацій в зарубіжних виданнях – 5.
http://orcid.org/0000-0002-8891-7002.

Місце роботи автора:

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 6033.
E-mail: olga.ovdiy@gmail.com

УДК 004.4'24 https://doi.org/10.15407/pp2019.01.048

Р.С. Шевченко, А.Ю. Дорошенко

TERMWARE-3 – СИСТЕМА ПЕРЕПИСУВАННЯ ТЕРМІВ, ЗАСНОВАНА НА КОНТЕКСТНОМУ ЧИСЛЕННІ

У статті описується конструкція системи переписування термів TermWare-3, що побудована на основі рефлексивного числення контекстних термів, коли структура терму включає до себе, окрім дерева, ще й внутрішній контекст та обмеження на співставлення при застосуванні (зовнішній контекст), що дозволяє занурити операції розв'язування імен в математичну семантику переписувальних правил. Описується метод ефективної диспетчеризації вибору правил, а також автоматичне перетворення виразів між системами на основі алгебраїчних типів мови Scala та контекстними термами.

Ключові слова: автоматизація розробки програмного забезпечення, мови програмування, переписування термів, системи типів, алгебра типів, аналіз коду, termware.

Вступ

Системи переписування та логічного виводу давно застосовуються для різних задач. Перша версія системи TermWare описана у статті [1] і з'явилась у 2003 році як система термів, заснована на безтиповому переписуванні термів [2], що доповнена взаємодією з базою фактів. Ця система довела свою ефективність для широкого кола задач [3, 4]. Друга версія, була заснована на тому ж математичному апараті, але на відміну від першої версії мала інший спосіб реалізації. Тоді як у першій версії рекурсивні алгоритми обходу дерева інтерпретувались за допомогою рекурсивних Java методів, то в другій – за допомогою вбудованого нерекурсивного інтерпретатора, що не використовує вбудований стек JVM для обходу дерев. Це дозволило застосування TermWare-2 в індустріальних задачах, де можна було подати кожен файл нетривіальної Java-програми як один великий терм і виконувати перетворення, що потребують обходу всіх файлів проекту. У такому вигляді система TermWare-2 застосовується і нині [3, 4].

Система TermWare-3 (і побудована на її основі мова програмування Vavilon), з одного боку, зберігають основні риси TermWare-2, як вбудованої високорівневої бібліотеки, з іншого – додають нові можливості застосування примітивів контекстного числення [5]. Як і попередня версія, TermWare-3 – це бібліотека, що містить в собі операції роботи з термами та можливість створення та застосування переписувальних правил до об'єктів користувача.

1. Основні конструкції TermWare-3

У цьому розділі ми приведемо основні конструкції числення контекстних термів та наведемо спосіб їх представлення у вигляді Scala-конструкцій TermWare-3.

Нехай T – множина термів, що складається з множини константних символів C, з виділеною підмножиною атомів $A \subset C$ та функціональних термів $F \subset A$. Побудуємо на її основі множину контекстних мультитермів $CT(T)$, що буде включати в себе:

- константи $C_i, i \in I$:

– константні терми, що не є атомами, представлені як трейти (характеристики) PrimitiveTerm[T], де T – це примітивний тип Scala, а PrimitiveTerm[T]; існує неявне перетворення між примітивними типами Scala та PrimitiveTerm;

– атоми, що представлені як трейти AtomTerm; існує неявне перетворення між Scala символами та AtomTerm;

- функціональні терми $f_i(t_1, \ldots, t_n) \in F, i \in I$. TermWare-3 підтримує більш застосовану до способу мислення розробників концепцію *структурних термів* $f_i(n_1 = t_1, \ldots, n_m = t_m)$, де n_1, $n_2 \ldots n_m$ – імена (атоми), t_1, ..., t_m – мультитерми; структурні терми, що є абстракцією виклику функції, що в сучасних мовах може

включати іменовані параметри та значення за замовчуванням. Ці структурні терми представлені як вигляди трейта StructuredTerm. Існує реалізація цього трейта для *case*-классів, головне в цьому трейті – операція іменування;

- перетворення (стрілки) $t_1 \to t_2$:

– реалізація Scala надає трейт ArrowTerm; його основна функціональність – можливість виділення лівої та правої частини перетворення, жодна з яких не є пустою;

– внутрішній контекст *context*(*a*,*b*) або $a@b$, що можна читати як a в контексті b, де a та b – будь-які терми (в формулах іноді використовується нотація a^b); в Scala реалізація будь-якого ординарного терма може мати внутрішній контекст (можливо пустий), для операцій з ним існують методи, описані в трейті ContextBase;

- зовнішній контекст a -: b що можна описати як: "a при умові контексту b". Будемо також інколи використовувати позначення b:-a, тобто, контекст має бути ближче до двокрапки. Ідея в тому, що при використанні a -: b як паттерна, його можна співставляти (як a) лише у випадку, коли контекст співставлення сумісний з b (детальніше описується при опису уніфікації). В реалізації Termware-3 будь-який терм має зовнішній контекст, який за замовченням є '*' – універсальним мультитерм, що сумісний з будь-яким термом;

- множинні мультитерми (або *or*-вирази); $\{t_1 \ \dots \ t_n\}$ (або t_1 *or* t_2 *... or* t_n) – можна розглядати як просто множину ординарних термів. Пуста множина еквівалентна виділеному терму EmptyTerm. В TermWare-3 множинний мультитерм представлений трейтом OrSet, що на додаток до звичайних операцій над термами, дає доступ до вибірки та переліку елементів. Також, над термами визначена операція *or*.

- сумісні множинні мультитерми (або *and*-вирази); $\ll t_1, \dots t_n \gg$ (або $t_1 and\ t_2, and \dots t_n$). Вони відрізняються від or-термів тим, що мають бути сумісні між собою. Формальне визначення сумісності наведемо після переліку термів, неформально – два терми сумісні, якщо це стрілки з різними лівими частинами або один з них є множинним термом або вони однакові. Пустий множинний мультитерм є універсальним термом. TermWare-3 надає трейт AndSet та операцію *and* над мультитермами;

- послідовність альтернатив $t_1 | \ t_2 | \ \dots | \ t_{other}$, або t_1 *orElse* t_2 *orElse* ... t_n , при співставленні результатом є перша можлива альтернатива. TermWare-3 надає операцію *orElse* над термами;

- захищений терм: *if(guard, t)*, де співставлення виконується лише якщо вираз *guard* інтерпретується в *true* у логічній системі, що може задаватись як параметр системи. У TermWare є трейт IfTerm;

- порожній мультитерм $\emptyset$, який можна інтерпретувати як "пустий множинний мультитерм", що у TermWare-3 представлений об'єктом EmptyTerm;

- універсальний мультитерм $*$, який можна інтерпретувати як "сумісний з будь-яким мультитермом", представленим об'єктом StarTerm.

Користуючись базисом цих елементів, можна емулювати конструкції стандартної логіки. Так, наприклад, підстановка може бути представлена як сумісний терм: $\ll x_1 \to t_1, \dots x_n \to t_n \gg$, де x_i – атоми з іменами, що відповідають іменам змінних, а t_n – результати. Додавання ще одного елемента або створить нову сумісну підстановку, або перетворить підстановку у пустий терм, що буде свідчити про неуспішність операції.

Вільні змінні з іменами x_i можуть бути емульовані як атоми з іменами та підстановкою, що перетворює ці імена в обмеження. Вираз

$$f(x_1, x_2)@\{\ x_1 \to *\ , x_2 \to *\ \}$$

відповідає вільним змінним x_1 та x_2 без обмежень. Для змінних з обмеженнями використовуються захищені терми:

$$f(x_1, x_2)@\{x_1 \to if\ (this < 2)\ *, x_2 \to$$

$$if\ \big(isString(this)\big) \to * \}.$$

При співставленні x_1, *this* поміщається у зовнішній контест терму і вираз *this* < *2* інтерпретується у базовій логіці.

Природним чином емулюється поняття типу: (*x*:*T*) це скорочення для захищеного терму *if* (*T.resolve*(*check*),*this*). Таким чином, тип у TermWare-3 є доведенням властивості безтипового виразу. Тобто останній вираз можна переписати як:

$$f(x_1,x_2)@\{x_1 \to if\ (this < 2) *, x_2 \to if\ (this{:}\ String) \to * \}.$$

Саму систему переписувальних правил також можна представити у вигляді терму: це буде сумісна множина стрілок

$$\ll\ p_1 \to t_1, \dots p_n \to t_n \gg,$$

де $p_1, \dots p_n$ – паттерни, $t_1, \dots t_n$ – відповідні підстановки. Як видно, підстановка є частинним випадком систем правил, і там основним є правило підстановки одного правила:

$$app\big((x \to y), z\big) = \begin{cases} subst\big(y, unify(x,z)\big), \\ unify(x,z) \neq \emptyset, \\ \emptyset, unify(x,z) = \emptyset. \end{cases}$$

Місце в системі правил – визначається операцією вибору. В такому представленні, переписувальні правила мають бути когерентними (тобто не суперечити одне одному).

Ще один варіант представлення – впорядкований набір правил, що може бути представлений за допомогою переліку альтернатив. Такий терм має вигляд:

$$x_1 \to t_1 | \ x_2 \to t_2 | \dots | \ x_n \to t_n.$$

Зазначимо, що якщо терми з більш-специфічним паттерном знаходяться раніше більш загальних, то ці два представлення еквівалентні [5].

За браком місця не будемо наводити повний опис формальної системи. Найбільш близький опис є у [5, 6]. Там подана попередня формалізація, де немає різниці між сумісними та несумісними множинами термів.

2. Імплементація

Стандартний метод представлення числення в Scala – моделювання основних конструкцій у вигляді ієрархії *case*-класів та побудова алгебри операцій над цими case-класами як окремого об'єкта. В TermWare-3 прийнятий інший підхід, що на перший погляд вступає у протиріччя з загальноприйнятим функціональним стилем. А саме, основні конструкції є трейтами, що допускають різні реалізації, а алгебра операцій представлена як методи цих трейтів. Основна причина такого дизайну – можливість ефективної реалізації.

Як приклад, подивимось на проблему диспетчерізації правил. Нехай у нас є терм *x* та система переписувальних правил:

$$\ll\ p_i \to y_i \dots \gg.$$

Як знайти правило з лівою частиною, що сумісне з *x*? Якщо ми використовуємо систему *case*-класів, то нам потрібно або організовувати пошук лівої частини в послідовності правил, що не є ефективним методом, або явно перетворювати систему правил в оптимізоване представлення, що ускладнює зовнішній інтерфейс бібліотеки [7].

Наприклад, розглянемо наступну систему правил:

$$\begin{pmatrix} f(x,g(y),x) \to p_1(x,y) | \\ f(x,x,x) \to p_2(x) \\ f(x,y,x) \to p_3(x,y) \\ g(y) \to y \\ x \to A \end{pmatrix} @ \begin{pmatrix} x \to * \\ y \to * \end{pmatrix}.$$

При співставленні раціонально тримати в пам'яті не список стрілок, а структуру, що застосована для швидкого метчінгу, показану на наступному рисунку.

Тут кожна вершина використовує табличний пошук (за арністю або ім'ям головного терма), далі виконується співставлення з типовим термом, і якщо воно

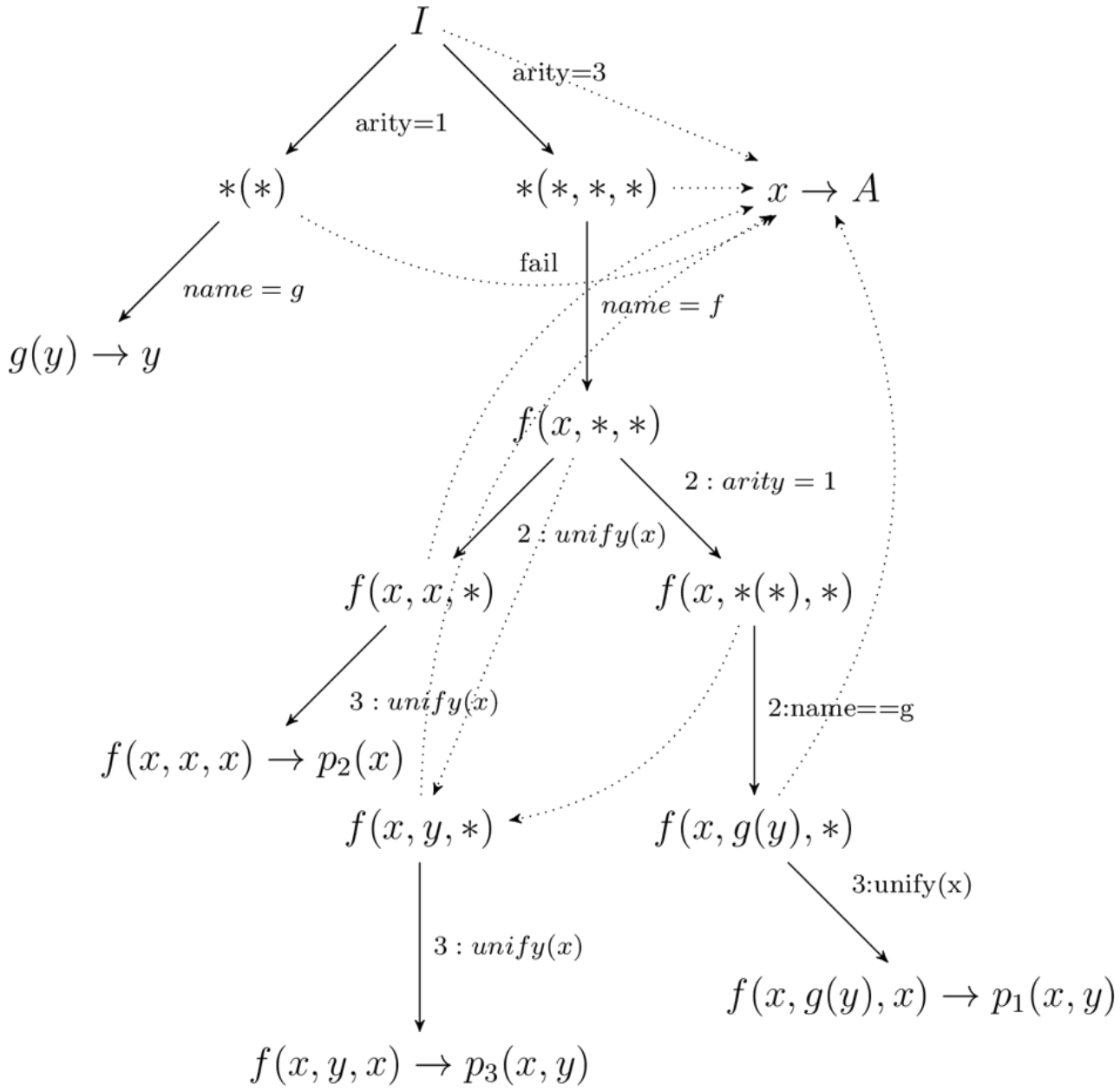


Рисунок. Структура для швидкого метчінгу

правильне – то проводиться остаточно перевірка, якщо ні – співставлення продовжується з “fallback”-вершини, перехід до якої показано пунктирною стрілкою.

Якщо взяти, наприклад, набір даних $p(1,1,1)$, то для визначення потрібного правила досить двох вибірок з таблиці: перший за арністю, другий – за іменем, а потім можна відразу переходити до правила $x \to A$; $f(1,1,1)$, що потребує трьох співставлень. У існуючій реалізації, відбувається спочатку табличний пошук за арністю, далі – табличний пошук за іменами і лише на третьому (передостанньому) рівні – повне співставлення. Теоретично ще можливо ввести кількісну оцінку складності співставлення та будувати структуру, що буде мінімізувати цю оцінку на наборі вхідних даних.

3. Приклад застосування

Наведемо приклад застосування аналізу контексту, на прикладі імплементації перевірки контрактів.

Нехай у нас є опис контракту на мові Solidity [8], що є типовим представником об’єктно-орієнтованих мов з вкладеними областями визначення. Для прос-

тоти викладу, ми не розглядатимемо фазу синтаксичного аналізу та припустимо, що об'єктом нашого розгляду є представлення Solidity AST у вигляді глибоко вбудованого (deep embedding) [9] виразу Scala.

Контракт на Solidity фактично є описом актора, що активується при надходженні транзакції, де може бути вказано додаткове повідомлення, що декодується у виклик методу. Він включає у себе опис стану актора, набір можливих функцій та вихідних випадків (events), що доступні для спостереження, разом з визначенням необхідних структур даних та модифікаторів.

На Scala це описується наступною ієрархією:

```
case class Contract(
      name: String,
     contractMember: Seq[ContractMember])

sealed trait ContractMember

case class VarDef[T](
  name:String,
  dataType: DataType[T],
  scope: Scope,
  location: DataLocation) extends ContractMember

sealed trait FunctionLikeMember extends ContractMember

case class ConstructorMember(
    params: Map[String,VarDef[_]],
    statements: IndexedSeq[Statement]
  ) extends FunctionLikeMember

case class FunctionMember(
  params: Map[String,VarDef[_]],
  statement: IndexedSeq[Statement])
```

Тут DataType – це тип даних, що може бути або вбудованим, або побудованим користувачем:

```
sealed trait DataType[T]

sealed trait Primitive[T] extends DataType[T]

case class IntPrimitive(value: Int) extends Primitive[Int]

case class AddressPrimitive(value: Long)
                    extends Primitive[Long]

 ……..
case class Struct(fields:
Map[String,DataType[_]])
              extends
DataType[Map[String,DataType[_]]]

case class Mapping[
           K:Primitive,
V:DataType](value:Map[K,V])
                          extends
DataType[Map[K,V]]

……
case class SArray[T:DataType](
              value: IndexedSeq[T]
       ) extends DataType[IndexedSeq[T]]
```

і нарешті Statement – це може бути виклик методу, присвоювання значення, ініціалізація локальної змінної тощо. Скорочено це можно проілюструвати наступним фрагментом:

```
case class MethodCall[O,T](obj: Expr[O],
params: LExpr[_]) extends Expr[T]

case class VarName[T](name:String)  extends
LExpr[T]

case class Assignment[T](left: LExpr[T] ,expr:
Expr[T]) extends Statement

case class VarInit[T](name:String,
        dateType:DataType[T],
        initExpr: RExpr[T]) extends Statement

case class IfStatement(condition:RExpr[Boolean],
```

```
        ifTrue: Statement,
        ifFalse: Statement) extends Statement
```

Як бачимо, вся програма представлена як алгебраїчна структура даних. В TermWare-3 реалізовано автоматичне перетворення екземплярів алгебраїчних типів у терми: все що потрібно, це проімпортувати DSL та викликати розширений метод перетворення:

```
val programTerm = ast.toTerm()
```

Але за замовчуванням, структура AST буде перенесена в таку ж структуру терму без побудови контекста. Для того, щоб показати, що деякі частини належать до контексту, потрібно вказати відображення поля в розташування підтерму (що може бути або в підтермі, або в контексті):

```
implicit object ContractContextPlacement
                extends AsContext[

Seq[ContractMember],
                              Symbol,
                              ContractMember]{

 override def map(in: Seq[ContractMember]):
     Try[Map[Symbol, ContractMember]] = {
  Try {
   in.map(m => (Symbol(m.name), m)).
                 groupBy(_._1).
                mapValues {
    s =>
     if (s.tail.nonEmpty) {
      throw new IllegalArgumentException(
              "duplicate name ${s.head._1}")
     } else {
      s.head._2
     }
   }
  }
 }

 override def unmap(in: Map[Symbol, Con-
tractMember]): Try[Seq[ContractMember]] = {
  Success(in.values.toSeq)
 }
}
```

І схоже перетворення потрібно вказати для VarDef, щоб визначення змінної потрапляло у контекст блоку. Як бачимо, задача цього перетворення – перетворення терму на маппінг, що згодом перетворюється на набір впорядкованих даних. Також зазначимо використання типу Symbol як ключа відображення, замість String. Це зроблено тому, що String за замовченням відображається у PrimitiveTerm[String] а Symbol – в AtomTerm.

Також має сенс імена змінних та частини селектори теж представляти як атоми:

```
implicit def varNameFieldMapping[T] =
  new AsFieldMapping[VarName[T],Symbol] {
 override def map(in: VarName[T]):
Try[Symbol] = Success(Symbol(in.name))

 override def unmap(in: Symbol):
Try[VarName[T]] = Suc-
cess(VarName(in.toString()))
}
```

Тепер *programTerm*, представлений як терм, в контексті якого знаходяться визначення (тут ми для простоти оминули додавання стандартних елементів програми, що визначені в специфікації Solidity). Ми можемо використовувати навігацію за символами: *x.resolve(name)* видасть нам визначення імені.

Розглянемо, для прикладу, програму перевірки (чекер) так званої «reentrance» [10] – помилки в контрактах, коли контракт робить транзакцію, не змінюючи свого стану. Саме ця вразливість була використана у відомому зломі DAO [11].

Приклад проблеми такого роду можна побачити в наступному коді:

```
contract Example {

  mapping (address=>uint) balances;

  ….

 function add() {
   balances[sender.address] += msg.valuel
```

```
 }

 function withdraw(uint amount) {
   if (balances[sender_address] > amount) {
      sender.address.send(amount);
      balances[sender.address] -= amount;
   }
}
```

Тут у функції *withdraw*, метод *send* викликається до модифікації *balances[sender.address]*. Оскільки send – це фактично виклик функції прийому платежа в іншому контракті, то інший контракт може викликати функцію *withdraw* ще раз, до того як поверне керування, і отримати ще один трансфер на свою адресу.

Система правил для знайдення цієї вразливості може виглядати так:

```
StateVars(statements, history) -> {

 statements . (seq(head,tail) ->
    IfStatement(expr,ifTrue,ifFalse) ->
                                Stat-
eVars(seq(ifTrue,tail)) &&
                                Stat-
eVars(seq(ifFalse,tail)) head
    |
     StateVars(tail, StateVars(head) + history)
   )

 Assignment(left,expr) -> StateVars(left)

 MethodCall(account,'send,params)
  if (resolve(account).type == address  &&
                                    history.isEmpty)
      -> Violation("reentrance-bug")
    |
   MethodCall(obj,metod,params) ->
     let v = re-
solve(obj).resolce(method).statements
      StateVars(tail,StateVars(v) + history)
```

Тут пропущені деталі, які не впливають на алгоритм, такі як вказання набору змінних. Цю систему правил ми маємо викликати до кожного методу. Тобто, ми проводимо рекурсивний спуск за послідовністю виразів, введемо в *history* множину і коли потрапляємо на виклик методу *send*, перевіряємо, що множина змін не є пустою.

Якщо ми зустріли виклик об'єкта в програмі, то додаємо до множини змін його слід (трейс).

Висновки

Отже, в даній роботі представлена система TermWare-3 на основі імплементації контекстного числення термів. Це дозволяє застосувати методи переписувальних правил до великих систем і подавати зв'язки між різними частинами системи у явному вигляді за допомогою розв'язання імен (резолвінгу).

Подальші кроки, що планується здійснити в цьому напрямку, це:

- адаптація застосування TermWare-3 у прикладних проектах;
- інтеграція з аналізаторами мов програмування Java та C, таким чином, щоб існувала можливість міграції систем на основі TermWare-2 на нову платформу;
- імплементація відображення системи типів Scala на TermWare-3 з тим щоб отримати типобезпечне вбудовування в Scala;
- побудова та аналіз ефективних імплементацій базових операцій.

Література

1. Shevchenko R., Doroshenko A. Managing Business Logic with Symbolic Computation. Information Systems Technology and Applications: Proc. 2-nd Intern. Conf. ISTA'2003, June 19–21, 2003. Kharkiv, Ukraine. P. 143–152.
2. Marc Bezem, Jan Willem Klop, Roel de Vrijer ("Terese"), Term Rewriting Systems ("TeReSe"), Cambridge University Press, 2003.

3. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. *Fundamenta Informaticae*. 2006. Vol. 72, N 1–3. P. 95–108.
4. Eugene Tulika, Anatoliy Doroshenko, Kostiantyn Zhereb. Using Choreography of Actors and Rewriting Rules to Adapt Legacy Fortran Programs to Cloud Computing. June 2017. Communications in Computer and Information Science.
5. Шевченко Р.С., Дорошенко А.Е. η-исчисление – реалистичная формализация класса переписывающих систем. Проблеми програмування. 2011. № 2. С. 3–11.
6. Шевченко Р.С. Числення контекстних термів для систем переписування. *Проблеми програмування.* 2018. № 2–3. С. 21–30.
7. Шевченко Р. Про імплементацію систем контекстних термів. *Winter InfoCom Advanced Solutions*. 2018. 41 с.
8. Ethereum Foundation. The solidity contract-oriented programming language. https://github.com/ethereum/solidity.
9. Jeremy Gibbons and Nicolas Wu. Folding domain-specific languages: deep and shallow embeddings (functional Pearl). In Proceedings of the 19th ACM SIGPLAN international conference on Functional programming (ICFP '14). 2014. ACM, New York, NY, USA. P. 339–347.
10. Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16). ACM, New York, NY, USA. P. 254–269.
11. "Report of Investigation Pursuant to Section 21(a) of the Securities Exchange Act of 1934: The DAO" (PDF). Securities and Exchange Commission. July 25, 2017.

References

1. Shevchenko R., Doroshenko A. Managing Business Logic with Symbolic Computation. Information Systems Technology and Applications: Proc. 2-nd Intern. Conf. ISTA'2003, June 19–21, 2003, Kharkiv, Ukraine. P. 143–152.
2. Marc Bezem, Jan Willem Klop, Roel de Vrijer ("Terese"), Term Rewriting Systems ("TeReSe"), Cambridge University Press, 2003, ISBN 0-521-39115-6.
3. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae. 2006. Vol. 72, N 1–3. P. 95–108.
4. Eugene Tulika, Anatoliy Doroshenko, Kostiantyn Zhereb. Using Choreography of Actors and Rewriting Rules to Adapt Legacy Fortran Programs to Cloud Computing. // June 2017. Communications in Computer and Information Science, DOI: 10.1007/978-3-319-69965-3_5
5. Shevchenko R.S., Doroshenko A.Y. η-calculus – Realistic Formalization of a Class of Rewriting Systems. Problems of Programming. 2011. N 2. P. 3–11.
6. Shevchenko R.S. A Calculus of Context Terms for Rewriting Systems. Problems of Programming. 2018. N 2–3. P. 21–30.
7. Shevchenko R. On Implementation of Term Systems. Winter InfoCom Advanced Solutions, 2018. 41 p.
8. Ethereum Foundation. The solidity contract-oriented programming language. https://github.com/ethereum/solidity.
9. Jeremy Gibbons and Nicolas Wu. Folding domain-specific languages: deep and shallow embeddings (functional Pearl). //In Proceedings of the 19th ACM SIGPLAN international conference on Functional programming (ICFP '14). 2014. ACM, New York, NY, USA, 339-347. DOI: https://doi.org/10.1145/2628136.2628138
10. Loi uu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, nd Aquinas Hobor. Making Smart Contracts Sarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). ACM, New York, NY, USA, 254-269. DOI:https://doi.org/10.1145/2976749.2978309
11. "Report of Investigation Pursuant to Section 21(a) of the Securities Exchange Act of 1934: The DAO" (PDF). Securities and Exchange Commission. July 25, 2017.

Одержано 08.02.2019

***Про авторів*:**

Шевченко Руслан Сергійович.
Підприємець.
Кількість наукових публікацій в українських виданнях – 13.
Кількість наукових публікацій в зарубіжних виданнях – 5.
http://orcid.org/0000-0002-1554-2019,

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу
теорії комп'ютерних обчислень,
професор кафедри автоматики та
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 Хірша – 5.
http://orcid.org/0000-0002-8435-1451,

Місце роботи авторів:

ПП "Руслан Шевченко".
E-mail: ruslan@shevchenko.kiev.ua

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.
E-mail: doroshenkoanatoliy2@gmail.com

УДК 004.853, 004.55 https://doi.org/10.15407/pp2019.01.057

Ю.В. Рогушина

ЗАСОБИ ТА МЕТОДИ АНАЛІЗУ НЕСТРУКТУРОВАНИХ ДАНИХ

Проаналізовано сучасні засоби аналізу неструктурованих даних (НСД) та вплив Big Data на актуальність цього напрямку досліджень. Розглянуто перспективи використання фонових знань для такого структурування. Обґрунтовано доцільність застосування для цього таких стандартів W3C, як RDF та OWL. Використання семантичних Wiki-технологій для створення розподілених інформаційних ресурсів не тільки дозволяє досить легко додавати структурування до НСД, але й є джерелом фонових знань для аналізу довільних природномовних текстів відповідної предметної області. Запропоновані в роботі моделі та методи дозволяють вдосконалити процес генерації таких знань.

Ключові слова: неструктуровані дані, Text Mining, онтологія, Semantic Web, Wiki.

Вступ

На даний час світовим співтовариством вже усвідомлений головний напрямок у боротьбі з інформаційним вибухом – перехід від збереження й обробки даних до накопичення й обробки знань. Тому виникає потреба у засобах та методах здобуття знань з тих даних, що генеруються в процесі діяльності людства та можуть бути корисними для подальшого використання. Актуальність проблеми загострюється через стрімке поширення Big Data, яке викликає потребу в нових, більш ефективних методах аналізу розподілених та гетерогенних даних.

Обробка великих обсягів інформаційних ресурсів різного походження та з наперед не відомими моделями даних (в такому випадку говорять про неструктуровані дані), для яких не придатні традиційні СКБД, потребує спеціалізованих засобів їх представлення та аналізу.

Ще у 1998 році аналітики з Merrill Lynch сформулювали емпіричне правило: біля 80 % – 90 % всієї потенційно корисної ділової інформації генерується в неструктурованій формі [1]. Прогнозується, що до 2025 року глобальна датасфера зросте до 163 зетабайт, і 70 % – 80 % її буде неструктурованою.

Визначення НСД

НСД – дані, для яких не визначені окремі елементи, їх властивості, можливі значення та спосіб їх кодування.

НСД – це інформація, яка не має попередньо визначеної моделі даних або не організована заздалегідь. Це призводить до проблем, пов'язаних з її зберіганням (традиційні БД не розраховані на таку невизначеність) та аналізом. Саме НСД потенційно мають найбільшу цінність як джерела нових знань: чим більше даних доступних для аналізу, тим точніші результати. Прикладами НСД можуть бути книги, журнали, документи, метадані, медичні записи, аудіо, відео, аналогові дані, зображення, файли та неструктурований текст, наприклад, тіло повідомлення електронної пошти, Web-сторінки або слова документ процесора.

Сьогодні у більшості випадків під НСД розуміють текстову інформацію – набори слів природної мови (ПМ) довільної довжини, поєднані за слабо формалізованими лінгвістичними правилами та представлені в електронній формі. Це пояснюється тим, що саме текстова інформація містить найбільш корисні для подальшого використання відомості. Такі НСД можуть містити також дати, числа тощо. Приклади текстових НСД:

- електронна пошта;
- ПМ-документи в різних форматах;
- відомості з соціальних мереж (YouTube, Facebook, Twitter, LinkedIn, Flickr тощо);
- дані з мобільних пристроїв (текс-

тові повідомлення й інформація про місце розташування) та Інтернету речей;

- контент Web-сайтів.

Найбільш поширені приклади НСД інших типів [2] – це потокове відео, інформація від супутників, дані радарів чи сонарів. Засоби аналізу таких НСД значно більш спеціалізовані.

Іноді досить складно відрізняти структуровані та НСД. Один з критеріїв визначення структурованості даних – для елемента таких даних можна створити синтаксичний аналізатор. Термін НСД не є точно визначеним з декількох причин [3]:

- структура може міститися у таких даних, але не мати формального визначення;
- дані, що мають певну структуру, можуть бути охарактеризовані як неструктуровані, якщо ця структура не є корисною для цілей їх обробки;
- неструктурована інформація може мати певну структуру (бути слабо структуровану або навіть структурованою), яка не може бути застосована для автоматизованої обробки без додаткових уточнень.

Таким чином, дані розглядаються як НСД у тих випадках, коли відомості про їх структуру не можуть зробити аналіз даних більш ефективним.

Неструктурована інформація може зберігатися у формі об'єктів (файлів чи документів), що самі мають структуру. Наприклад, тіло листа або вкладення до електронної пошти – це неструктуровані дані, але їх місцезнаходження в пошті задається її структурою. Сполучення структурованих і неструктурованих даних у сукупності також є НСД.

Властивості НСД

НСД, на відміну від структурованих даних, які здебільшого не мають антропогенних особливостей, досить часто створюються безпосередньо людьми, і тому системи обробки та аналізу НСД мають враховувати «людський фактор».

Властивості НСД:

- *гетерогенність*. Для НСД існує величезна кількість різних способів створення, джерел інформації та причин, через які ці дані не можуть бути структуровані і поміщені в будь-яку СКБД, а лише у файли різноманітних форматів (приклад – наукові статті мають певну структурованість та обов'язкові елементи, але їх неможливо представити інакше, як файли текстових редакторів);
- *неоднозначність*. Висловлення двох осіб, що збігаються дослівно, можуть мати різний зміст у залежності від досвіду, поглядів тощо, а та сама ідея може бути виражена різними словами (наприклад, твердження експерта "я не зрозумів цю статтю свідчить про низьку якість статті, а те саме твердження студента – про його низьку освіту);
- *контекстна залежність.* Те саме слово чи ім'я можуть у різних умовах інтерпретуватися по-різному ("модель" у техніці та у математиці мають різне значення);
- *динаміка значення.* Слова можуть дуже швидко змінювати свій зміст, наприклад, назва нікому раніше не відомого населеного пункту через події, що відбувалися в ньому, може стати загальновідомою та отримати додаткове значення;
- *етнокультурна залежність.* У різних етносах і культурах, що використовують ту саму мову, слова можуть набувати різного сенсу і позначати зовсім різне.

Такі технології, як Data Mining, обробка природної мови і Text Mining, надають різні методи для пошуку структури в НСД. Загальні методи структуризації тексту зазвичай включають у себе ручну розмітку метаданими або тегами для подальшого структурування. Стандарт архітектури керування неструктурованою інформацією (Unstructured Information Management Architecture – UIMA) надає загальну основу для обробки цієї інформації для здобуття сенсу та створення структурованих даних.

Історія виникнення аналізу НСД

Найбільш ранні дослідження Business Intelligence (BI) зосереджувалися саме на неструктурованих текстових даних, а не на числових даних [4]. Проте ли-

ше на початку 21 століття технології наздогнали наукові дослідження. Поява Big Data наприкінці 2000-х років викликала підвищений інтерес до застосування неструктурованих аналітичних даних.

У 80-х і 90-х роках 20 ст. бізнес-аналітика (OLAP, інтелектуальний аналіз даних, ETL та сховище даних) була зорієнтована на структуровані числові дані, що зберігалися в реляційних базах даних.

Виділення аналізу НСД (UDA – unstructured data analysis) в окремий науково-технічний напрямок датується початком 2000 років, коли аналітики Gartner опублікували інформацію про високі затрати часу та праці на обробку даних – рутинна, не автоматизована робота з контентом займала до половини робочого часу. Незручність була пов'язана саме з необхідністю обробки текстових НСД у різних форматах: електронних листів, службових записок, новин, чатів, звітів, маркетингових матеріалів, презентацій тощо, які не можливо було занести до реляційних СКБД (деякі з таких даних є слабо структурованими або квазіструктурованими та супроводжуються метаданими – автор, місце створення, розмір – які можна помістити до СКБД).

На сьогоднішній день НСД складають найбільшу частку даних, що зберігаються (понад 80 % усіх збережених даних, а їхня кількість зростає на порядок швидше в порівнянні з структурованими даними), тому методи та засоби їх використання швидко розвиваються. Ці методи спрямовані на перетворення цих даних на структуровану інформацію, яка може використовуватися різними способами.

Text Mining як окремий напрямок з'явився наприкінці 1990-х років 20 ст. Ранні підходи розглядали текст як "мішок слів" (“bag of words”), таких як абревіатури, множини і сполучення, а також терміни з декількох слів, відомі як n-грами. Основний лексичний аналіз може враховувати частоти слів і термінів для виконання елементарних функцій, таких як спроби класифікувати документи за темами. Але не було можливості зрозуміти семантику документів. Нині Text Mining шукає приховані відношення та інші складні структури в наборах текстових даних.

Text Mining як основа аналізу неструктурованої текстової інформації

Аналіз текстових даних як технологія базується на лінгвістиці та інтелектуальному аналізі даних, що спочатку застосовувалися в аналітиці для розпізнавання в тексті особистих і географічних назв, дат, телефонних номерів та адреси електронної пошти. Більш складні методи дозволяли знаходити поняття і відношення між ними та навіть настрої.

Додання структури до НСД є складною науковою проблемою, якій приділяють увагу науковці на протязі довгого часу [5]. Актуальність проблеми збільшилася з поширенням Big Data. У найбільш узагальненому вигляді розв’язок проблеми пов’язують з побудовою розміченого графу, що відповідає вмісту НСД, та із співставленням таких графів. Інший аспект цієї проблеми пов’язують із знаходженням релевантних знань, з якими співставляють НСД.

Методи Data Mining включають класифікацію, кластеризацію, аналіз зв'язків, дерева рішень тощо [6]. Інтелектуальне моделювання використовується для таких бізнес-функцій, як оцінка кредитів, виявлення ризиків, виявлення шахрайства та прогнозування для прогнозування тенденцій залежної від часу інформації. Всі ці методи Data Mining можуть бути пристосовані до даних, отриманих з текстових джерел – наприклад, необхідно знизити високу розмірність текстової інформації. Дослідники використовують для вирішення цих питань статистичні методи (такі як розкладання сингулярних значень і векторні машини підтримки для зменшення розмірності) у поєднанні з алгоритмами машинного навчання (деревами рішень, нейронними мережами тощо) і більш глибокою лінгвістикою, що підтримує такі функції, як використання контексту для визначення семантичної неоднозначності.

Text Mining можна визначити як процес здобуття знань з колекції ПМ-документів за допомогою набору ін-

струментів для їх аналізу [7]. Аналогічно до Data Mining, засоби Text Mining прагнуть здобути з даних потрібну для діяльності користувача інформацію. У випадку Text Mining джерела даних – це колекції документів, і цікаві для користувачів шаблони потрібно знайти не серед формалізованих записів бази даних, а в неструктурованих текстових даних у документах цих колекцій.

Text Mining можна розглядати як окремий випадок Data Mining. Тому не дивно, що системи Text Mining та Data Mining мають багато подібностей в архітектурі. Наприклад, обидва типи систем використовують процедури попередньої обробки, алгоритми виявлення шаблонів і засоби візуалізації результатів для покращення перегляду наборів відповідей. Text Mining використовує багато специфічних типів моделей у своїх основних операціях виявлення знань, які були впроваджені та перевірені в дослідженнях Data Mining.

Оскільки Data Mining припускає, що дані зберігаються у структурованому форматі, попередня обробка в ньому фокусується на задачах очищення та нормалізація даних і створення великої кількості об'єднань таблиць. На відміну від цього, для Text Mining операції з попередньої обробки пов’язані з ідентифікацією та пошуком репрезентативних властивостей для документів, поданих природною мовою (ПМ). Ці операції попередньої обробки забезпечують перетворення НСД, що зберігаються в колекціях документів, в більш чітко структурований проміжний формат. Тому Text Mining також спирається на досягнення в інших дисциплінах, пов’язаних з обробкою природної мови: методи інформаційного пошуку, здобуття інформації та комп’ютерної лінгвістики на основі корпусу (рис. 1).

Для продуктивного здобуття корисних відомостей з даних, що містять «людську інформацію», крім пошуку, застосовують технології Text Mining, спеціалізовані на обробці ПМ. Уперше термін Text Mining було використано в 1995 році як альтернатива терміну «здобуття знань з тексту» (Knowledge Discovery from Text, KDT).

Складові Text Mining

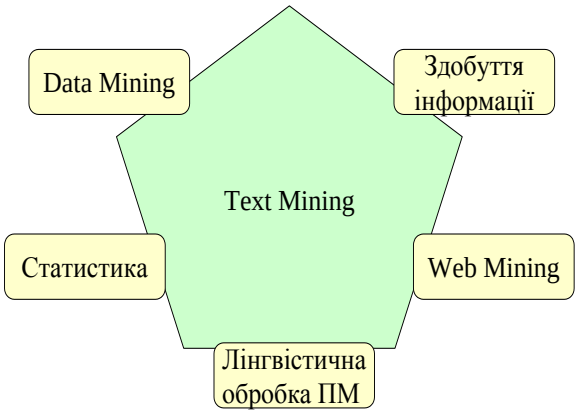


Рис. 1. Складові Text Mining

Text Mining має забезпечити перехід від НСД до структурованих з наступним аналізом. Найчастіше в цьому процесі ігнорується велика частина специфічних особливостей ПМ, які застосовуються тільки на попередньому етапі розбору текстів, а на наступних використовується модель «мішка слів», у якій не важливий порядок слів.

Emanu Text Mining. Потреба в технологіях Text Mining загострилася, коли кількість текстів стала перевищувати можливості сприйняття людиною та виникла потреба в автоматизації здобуття їх змісту. На рис. 2 показана узагальнена схема процесу Text Mining. На етапі попередньої обробки НСД перетворюються в структуровану інформацію, в якої потім виділяються істотні ознаки – атрибути та здійснюється їх дослідження.

Автори книги «Вступ до неструктурованих даних» (2007) (“Tapping into Unstructured Data”) Бів Інмон та Ентоні Несвіч, аналізуючи зв’язок між Business Intelligence та Text Mining у другій частині «Інтегрування неструктурованих даних у текстову аналітику і BI» (“Integrating Unstructured Data and Textual Analytics into Business Intelligence”), поділяють Text Mining на два напрямки: «виявлення» (Discovery) – дедуктивні методи підтвердження або спростування гіпотез та «аналіз» (Analysis) – статистика, кластеризація тощо.

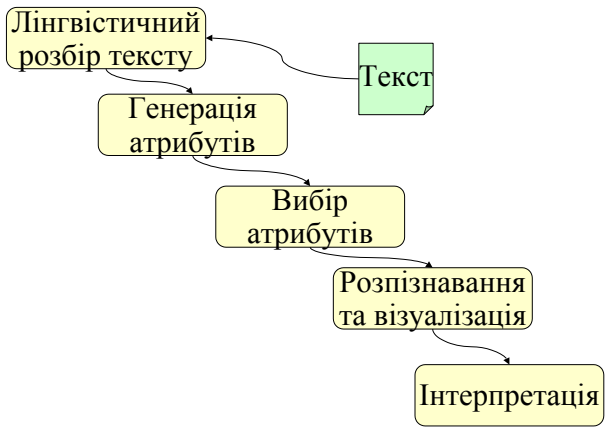


Рис. 2. Етапи Text Mining

Основні елементи Text Mining. Ключовим елементом Text Mining є колекція документів. У найпростішому випадку це довільна група текстових документів. Більшість рішень Text Mining спрямовані на виявлення моделей (шаблонів) у дуже великих колекціях документів, у тому числі – у сховищах Big Data.

Колекції документів можуть бути *статичними*, тобто початковий набір документів залишається незмінним, або *динамічними*, тобто до початкового набору можуть додаватися нові документи, а існуючі – оновлюватися.

Якщо колекція документів має великий розмір та швидко змінюється, то ручні спроби її аналізу не є ефективними. Автоматичні методи виявлення та вивчення взаємозв'язків між документами різко підвищують швидкість та ефективність дослідницької діяльності, але їх неможливо застосовувати на непідготовлених колекціях документів.

Документ – ще один основний елемент Text Mining. Це одиниця дискретних текстових даних в колекції, може корелюватися з деякими документами реального світу, такими як звіт, електронна пошта, стаття, прес-реліз або новини.

Документ може одночасно входити до різних колекцій документів або різних підмножин однієї колекції.

Елементи структурування текстових документів. Такі НСД, як текстовий документ, з деяких точок зору можна розглядати як структурований об'єкт. Наприклад, з лінгвістичної точки зору кожен документ містить велику кількість семантичної та синтаксичної структури, яка прихована в тексті. Крім того, елементи розмітки, такі як знаки пунктуації, великі літери, числа та спеціальні символи, та елементи форматування (таблиці, стовпці, параграфи тощо) можуть розглядатися як мова «м'якої розмітки», що допомагає визначити важливі підкомпоненти документів – назву, імена авторів, підрозділи тощо. Послідовність слів також може бути структурно значущим виміром документа. Крім того, деякі текстові документи можуть містити вбудовані метадані у вигляді формалізованих тегів розмітки, які автоматизовано генеруються текстовими редакторами.

Документи, які мають відносно мало таких елементів структурування (наприклад, наукові публікації та бізнес-звіти), називають *вільно форматованими (free- format*) або *слабо структурованими* (*weakly structured*). Документи з відносно більшою кількістю елементів структурування (наприклад, електронна пошта, Web-сторінки HTML) називають *частково структурованими* (*semistructured*) .

Операції попередньої обробки дозволяють використовувати в Text Mining багато різних елементів, що містяться в ПМ-документі для його перетворення з НСД з неявним структуруванням в явно структуровані дані. Однак, з огляду на потенційно велику кількість слів, фраз, речень та елементів форматування, які може мати навіть невеликий документ, навіть не враховуючи потенційно велику кількість різних значені, які кожен із цих елементів може мати в різних контекстах і комбінаціях, найважливішим завданням для більшості систем Text Mining є ідентифікація спрощеної підмножини властивостей (ознак) документів. Такий набір ознак називають *репрезентативною моделлю* документа: окремі документи характеризуються за допомогою наборів ознак, які містять їхні репрезентативні моделі. Але слід враховувати, що навіть у найбільш ефективних репрезентативних моделях кожен окремий документ у колекції має надзвичайно велику кількість властивостей.

Тому проблеми, пов'язані з високою розмірністю характеристик (тобто розміром і масштабом можливих комбінацій значень ознак для даних), зазвичай мають значно більше значення в системах Text Mining, ніж у класичних системах Data Mining.

Структуровані представлення ПМ-документів мають набагато більшу кількість потенційно репрезентативних ознак – і, отже, більшу кількість можливих комбінацій їх значень – ніж в реляційних або ієрархічних базах даних. Наприклад, у відносно невеликій колекції з 10–15 000 документів, можна виявити більше 25 000 нетривіальних слів. Навіть якщо працювати з більш оптимізованими типами властивостей, десятки тисяч ознак, пов’язаних з різними поняттями. можуть бути актуальними для однієї предметної області (ПрО). Кількість атрибутів у реляційній базі даних, які аналізуються в задачі інтелектуального аналізу даних, зазвичай значно менше. Висока розмірність потенційно репрезентативних властивостей спонукає до попередньої обробки тексту, спрямованої на створення спрощених моделей подання.

Ще однією характеристикою ПМ-документів є *розрідженість властивостей* (*feature sparsity)* – лише невелика частка всіх властивостей, можливих для колекції документів у цілому, з'являється в кожному окремому документі, і, таким чином, коли документ представляється у вигляді бінарного вектора ознак, майже всі значення вектора дорівнюють нулю.

Розмір кортежу також розріджений. Тобто деякі функції часто з'являються лише в декількох документах, а це означає, що підтримка багатьох моделей досить низька.

Властивості окремого ПМ-документа – це символи, слова, терміни і поняття. Оскільки алгоритми Text Mining обробляють представлення документів через набір властивостей, а не безпосередньо самі документи, виникає потреба у компромісі між двома важливими цілями.

Перша ціль полягає у тому, щоб досягти правильної класифікації обсягу і семантичного рівня властивостей для точного відображення значення документа в процесі виконання операції попередньої обробки тексту. Друга ціль – вибрати таке визначення властивостей, що є найбільш обчислювально ефективним і практичним для виявлення шаблонів. Такий вибір може підтримуватися валідацією, нормалізацією або посиланням на властивості з контрольованих словників або зовнішніх джерел знань, таких як словники, тезауруси, онтології або бази знань, щоб допомогти у створенні менших наборів властивостей з більшою семантичною значимість.

Хоча для представлення ПМ-документів можна використовувати багато потенційних властивостей, найчастіше використовуються такі чотири типи.

- *Символи*. Букви, цифри, спеціальні символи та пробіли є будівельними блоками семантичних ознак вищого рівня, таких як слова, терміни та поняття. Представлення на рівні символів може включати повний набір всіх символів для документа або деякого фільтрованого піднабору. Представлення на основі символів без інформації щодо позицій (тобто підходи з “мішком символів” – “bag-of-characters “) зазвичай мають дуже обмежену корисність для Text Mining. Представлення, які включають певний рівень позиційної інформації (наприклад, біграми або триграми) дещо корисніші.

- *Слова*. Конкретні слова, вибрані безпосередньо з ПМ-документа, є базовим рівнем для семантики. Одна властивість на рівні слів повинна мати значення не більше одного лінгвістичного маркера. Фрази та багатослівні вирази не складають окремих властивостей на рівні слів. Представлення документа на рівні слів може включати в себе ознаки для кожного слова в цьому документі, тобто текст документа представляється повним набором властивостей рівня слова. Це може призвести до того, що деякі представлення колекцій документів на рівні слів містять десятки або сотні тисяч унікальних слів у своєму просторі ознак. Проте, більшість представлень документів на цьому рівні демонструють принаймні деяку мінімальну оптимізацію і тому складаються з підмножин репрезентативних властивостей, які фільтруються

від таких елементів, як стоп-слова, символи та беззмістовні числа.

- *Терміни* – це окремі слова та багатослівні фрази, вибрані безпосередньо з корпусу вихідного документа за допомогою методології вилучення термінів. Функції на рівні термінів, у сенсі цього визначення, можуть бути складені тільки з конкретних слів і виразів, знайдених у рідному документі, для якого вони мають бути загалом репрезентативними. Отже, представлення документа на основі термінів обов'язково складається з підмножини термінів у цьому документі. Наприклад, якщо документ містив речення. Існують різні методології видобування термінів, які можуть конвертувати необроблений текст документа в послідовність нормалізованих термінів (токенізованих і лематизованих форм слова), помічених тегами відповідних часток мови. Іноді для нормалізації термінів також використовується зовнішній лексикон для забезпечення контрольованого словника. Методики видобуття термінів використовують різні підходи для генерування та фільтрації списку найбільш значущих термінів документа з цього набору нормалізованих термінів.

- *Поняття* – це властивості, створені для документа за допомогою різних методик категоризації. Властивості рівня понять можуть бути створені для документів вручну, але тепер частіше видобуваються з документів за допомогою складних процедур попередньої обробки, які ідентифікують окремі слова, багатослівні вирази, цілі речення або навіть більші синтаксичні одиниці, які потім відносяться до конкретних ідентифікаторів понять.

Багато методологій категоризації включають ступінь перехресного посилання на зовнішнє джерело знань; для деяких статистичних методів цим джерелом може бути просто анотована колекція документів. Для категоризації вручну і на основі правилах перехресні посилання і перевірка перспективних властивостей на рівні понять зазвичай включають взаємодію з зовнішніми БЗ, таким як існуюча онтологія домену, лексика або ієрархія формальних. На відміну від властивостей на рівні слів і термінів, властивості документа на рівні понять можуть складатися з слів, які не містяться у цьому документі.

З чотирьох типів описаних тут ознак терміни та поняття відображають властивості з найбільш виразними рівнями семантичної значущості, тому існує багато переваг для їх використання для представлення документів в Text Mining.

Що стосується загального розміру наборів властивостей, то представлення на основі термінів і понять мають приблизно однакову ефективність, але в цілому набагато ефективніші, ніж моделі документів на основі символів або слів. Представлення на рівні термінів легше згенерувати автоматично з тексту, ніж представлення на рівні понять. Проте представлення на рівні понять набагато корисніше для обробки синонімії та полісемії.

Представлення на основі понять дозволяють використовувати дуже складні ієрархії понять і різноманітні знання про домен, що надаються онтологіями та базами знань. Але представлення на рівні понять мають кілька потенційних недоліків: а) відносна складність застосування евристик під час операцій попередньої обробки, б) залежності багатьох понять від домену.

Використання фонових знань в Text Mining

У системах Text Mining поняття належать не тільки до дескриптивних атрибутів певного документа, а й до доменів (ПрО). ПрО у Text Mining – це спеціалізована область інтересів, для якої можуть бути розроблені спеціальні онтології, лексикони та таксономії.

Системи Text Mining можуть використовувати інформацію з формалізованих зовнішніх джерел знань для цих ПрО, щоб покращити попередню обробку документів та виявлення знань.

Знання ПрО (інша поширена назва – *фонові знання* (*background knowledge*)), можуть бути використані в Text Mining для попередньої обробки для поліпшення здобуття понять. Доступ до фонових знань – хоча і не є абсолютно необхідним для створення ієрархій концепцій в контексті

єдиного документу або збору документів – може відігравати важливу роль у розробці більш значущих, послідовних і нормалізованих ієрархій концепцій.

Text Mining використовує фонові знання більшою мірою Data Mining: властивості не є просто елементами в плоскому наборі, як це часто буває у структурованих даних, тому що вони пов'язуються за допомогою лексиконів і онтологій для підтримки розширених запитів.

Незважаючи на те, що операції попередньої обробки Text Mining відіграють важливу роль у перетворенні неструктурованого вмісту необробленої колекції документів у більш сприйнятливе представлення даних на рівні понять, основна функціональність систем Text Mining полягає в аналізі моделей *спільного виникнення* понять (*"concept co-occurrence"*) в документах колекції. В Text Mining використовуються алгоритмічні та евристичні підходи для розгляду розподілів, наборів, що часто повторюються *("frequent sets"*), та різних асоціацій понять на міждокументному рівні з метою надання користувачеві можливості виявити природу та взаємозв'язки понять, що відображені у колекції в цілому.

Наприклад, у колекції новин велика кількість статей, де йдеться одночасно про подію X та компанію Y, а також статей, де йдеться одночасно про компанію Y та продукту Z, може вказувати на інтерес до зв'язку між X та Z, хоча цей зв'язок не присутній у жодному документі.

У класичному Data Mining фонові знання із зовнішніх джерел використовуються для обмеження пошуку.

Системи Text Mining можуть використовувати інформацію з зовнішніх джерел знань в операціях попередньої обробки текстів і перевірки понять. Крім того, доступ до фонових знань може відігравати важливу роль у розробці змістовних, послідовних і нормалізованих ієрархій понять.

Додаткові знання, крім того, можуть бути використані іншими компонентами системи видобування тексту. Наприклад, одним з найбільш важливих застосувань фонових знань є побудова значущих обмежень для операцій виявлення знань. Аналогічно, фонові знання можуть також використовуватися для формулювання обмежень, які дозволяють користувачам підвищувати гнучкість при перегляді великих наборів результатів або при форматуванні даних для презентації.

Системи Text Mining можуть використовувати фонові знання, представлені у вигляді онтологій ПрО, що описує сукупність всіх важливих для ПрО фактів, класів і відношень між цими класами. Її можна розглядати як словник, побудований таким чином, щоб бути одночасно зрозумілим для людей і придатним для машинної обробки. Онтологія дозволяє визначити відношення часткового порядку між поняттями ПрО.

Один з прикладів онтології, що застосовується в Text Mining, – WordNet. Це розробка Принстонського університету для моделювання ПМ.

Системи розробки тексту також використовують фонові знання, що містяться в лексиконах ПрО. Цей термін близький до поняття тезаурусу.

Лексикон ПрО для онтології О – це кортеж

$$Lex = < S_C, \mathrm{Re}\, f_C >,$$

що складається з множини S_C, елементи якої – назви понять ПрО, а відношення $\mathrm{Re}\, f_C \subseteq S_C \times c$ лексичне посилання для понять, для яких $(c,c) \in \mathrm{Re}\, f_C$ виконується для всіх $c \in C \cap S_C$.

На основі $\mathrm{Re}\, f_C$ можна визначити, що для $s \in S_C$

$$\mathrm{Re}\, f_c(s) = \{c \in C \mid (s,c) \in \mathrm{Re}\, f_C\}.$$

Лексикон, подібний до WordNet, може служити точкою входу для фонових знань. Використовуючи лексикон, система Text Mining може нормалізувати ідентифікатори концепції, доступні для анотування документів у його корпусі під час попередньої обробки. Це дозволяє підтримувати за допомогою онтології, пов'язаної з лексиконом, такі операції, як вирішення синонімії, так здобуття інформації про семантичні відношення між поняттями. Крім то-

го, фонові знання дозволяють задавати параметри (значення атрибутів певного поняття) для пошукового запиту щодо екземплярів цього поняття, та визначати їх взаємини з екземплярами інших понять. Наприклад, можна шукати компанії, визначивши значення таких атрибутів, як продукція та місцезнаходження, або шукати компанії, місцезнаходження яких відноситься до класу “Столиця” .Такі атрибути та відношення мають бути доступні користувачеві у списку вибору при формуванні конкретного запиту. Крім того, це дозволяє визначити у запиті те відношення між поняттями, яке задовольняє користувачів. Наприклад, це дозволяє відокремити покупців продукту X від продавців цього продукту.

Моделі подання структурованих даних та їх використання для НСД

В роботі [8] пропонується опис простору даних, який дозволяє класифікувати моделі даних та засоби їх обробки.

Простір даних DS=<DB, DW, ODW, Wb, Nd, Gr, Int, Se, Wo, EM> – це множина даних з різними моделями подання. До таких моделей авторка відносить бази даних DB, сховища даних DW, статичні Web-сторінки Wb, НСД Nd, мультимедійні дані Gr, локальні сховища ODW, а також засоби інтеграції Int, пошуку Se та обробки Wo, що об’єднані середовищем управління моделями (EM).

Ці моделі даних ієрархічно впорядковані відповідно до їх виразної потужності: реляційна, багатовимірна, об’єктно-реляційна моделі, розширена мова розмітки інформації (Extensible Markup Language – XML) зі схемою, середовище опису ресурсів (Resource Description Framework – RDF), стандартний засіб опису зв’язків між об’єктами даних – онтології, описані за допомогою Web Ontology Language – OWL, структурований текст, неструктурований текст (рис. 3). Кожен учасник простору даних підтримує деяку модель даних і деяку мову запитів, відповідну до цієї моделі.

Документи та Web-сторінки можуть розглядатися в такому випадку як НСД. Процес розміщення таких інформаційних джерел у певній таксономії пов’язаний з їх класифікацією. Застосування стандартів W3C та онтологічного аналізу може застосовуватися для додавання структури до НСД.

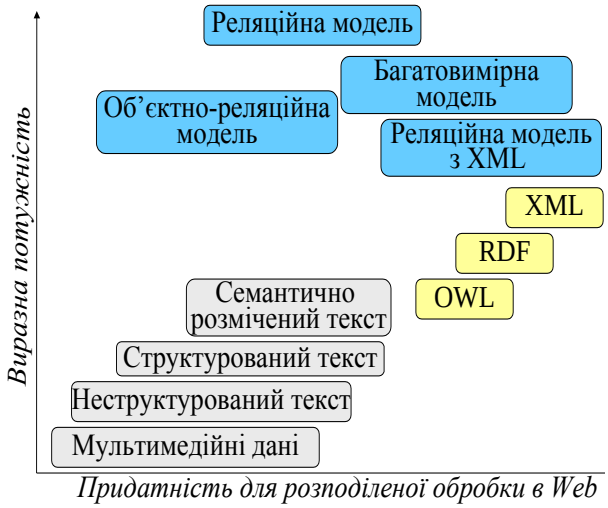


Рис. 3. Моделі подання даних

Найбільш розповсюдженою моделлю збереження структурованих даних з кінця 70-х років 20 ст. є реляційна модель, а стандартом на їхню обробку – мова SQL. Однак для НСД ця модель не ефективна.

Нереляційні моделі даних. Сьогодні задачі, що виходять за рамки реляційної моделі, прийнято відносити до класу NoSQL (звичайно розшифровується як Not Only SQL), кожен підклас якого вирішує окрему проблему, що погано реалізується за допомогою SQL, – наприклад, документо-орієнтовані, об'єктні та графові БД. Такі БД мають певні обмеження на операції, що підтримуються традиційними БД. Наприклад, великі розподілені БД повністю відмовляються від транзакцій, що забезпечує підвищення продуктивності за рахунок використання паралелізму.

Реалізована в проекті Hadoop технологія роботи з даними докорінно відрізняється від традиційних реляційних СКБД, призначених для роботи зі структурованими даними. NoSQL – сімейство технологій роботи з даними, які відрізняються від традиційних реляційних СКБД за наступними ознаками [9]: відсутність підтримки мови структурованих запитів

SQL; робота з неструктурованими чи слабко структурованими даними; відсутність механізмів забезпечення цілісності даних у тому вигляді, як вони реалізовані в класичних СКБД; розподілена реалізація з широкими можливостями горизонтального масштабування. У цілому основне призначення NoSQL полягає у можливості обробки великої кількості неструктурованих даних за нерегламентований час, але з гарантованим результатом. У цьому складається принципова відмінність NoSQL від традиційних СКБД, які забезпечують збереження інформації в чітко структурованому вигляді і гарантують час виконання операцій.

RDF як модель даних. Великий клас задач, які важко розв'язувати на реляційній моделі, – це задачі на сильно зв'язаних даних (графові задачі). Для них сьогодні найбільше поширення одержали RDF-сховища, які використовують стандарти W3C для мови RDF (Resource Description Framework) і запити SPARQL [10].

Основа RDF – це представлення даних у вигляді тверджень-трійок "суб'єкт-предикат-об'єкт". Для ідентифікації суб'єктів, об'єктів і предикатів використовується ідентифікатор URI (Uniform Resource Identifier), що є узагальненням поняття URL. Крім того, для подання об'єктів можуть використовуватися літерали.

На відміну від реляційної моделі, модель RDF досить гнучка – кожен суб'єкт може містити свої власні предикати й об'єкти, наприклад, у єдиній базі товарів усі товари мають предикат «Ціна», але в той же час холодильники можуть мати предикат «Обсяг морозильної камери», а телевізори – предикат «Діагональ екрана».

Модель RDF описує орієнтований граф, у якому кожна трійка – це опис зв'язку між двома вузлами.

Модель RDF служить для опису даних, але не описує методів їхньої обробки. Існує багато мов запитів до RDF: DQL, N3QL, R-DEVICE, RDFQ, RDQ, RDQL, SeRQL і т. д., але найпоширенішою є SPARQL – стандарт W3C, який, на відміну від SQL з неоднозначною граматикою і семантикою, має чітку структуру і більшу виразність. Основна частина запиту на SPARQL – шаблон, що описує підграф, який потрібно знайти в графі RDF. Цей шаблон представляється у вигляді набору трійок з перемінними. На сьогоднішній день SPARQL є однією з найбільш виразних мов обробки даних. Крім мови запитів, стандарт SPARQL регламентує протокол взаємодії з базою даних і формат результату, що є великим кроком вперед у порівнянні з SQL.

Рівень стандартизації RDF і SPARQL набагато вище, ніж у SQL, – зусиллями комітету W3C визначені стандарти не тільки на модель RDF і мову SPARQL, але і на ідентифікацію ресурсів (URI), протокол взаємодії компонентів (HTTP), точку доступу SPARQL тощо. Завдяки стандартизації дані з будь-якого RDF-сховища можна завантажувати в RDF-сховища різних виробників. Запити на SPARQL однаково виконуються на різних сховищах. У RDF легко зберігати метадані. На основі метаданих можна робити складні запити, вибираючи, скажемо, дані з конкретних джерел, у конкретному часовому діапазоні тощо.

Сьогодні спостерігається бурхливий розвиток ринку засобів розробки на основі моделі RDF. Деякі з них мають спеціалізовану архітектуру для обробки графів, а інші побудовані поверх реляційних БД. Найбільш поширені з них.

- *Apache Jena* – Java API для розробки застосунків Semantic Web, що містить кілька сховищ даних: Jena TDB – сховище RDF-трійок, Jena SDB – інтерфейс до реляційного сховища, In-Memory – сховище в пам'яті.
- *Ontotext OWLIM* – сімейство семантичних RDF-репозиторіїв з власним ядром, реалізованим на Java, з підтримкою семантики на RDFS (RDF Scheme) і OWL.
- *OpenLink Software Virtuoso* з власним RDF-сховищем, повною реалізацією SPARQL та можливістю читання RDF з файлів формату XML і Turtle.

Великі корпорації, такі як IBM і Oracle, також розробляють власні RDF-рішення. IBM пропонує NoSQL Graph Support, з інтерфейсом на основі розширення API Jena. Oracle у Spatial and Graph

Option підключила RDF до засобу обробки просторових даних Spatial Data Option.

RDF-сховища дозволяють збирати, зберігати й індексувати дані з різних джерел – зокрема, при рішенні актуальної задачі інтеграції сервісів, що зводиться до об'єднання розрізнених реляційних БД у єдину базу і приводить до задачі обробки квазіструктурованих даних. Дані усередині кожної з таких БД строго структуровані для роботи з реляційною моделлю, але кожна база структурована по-своєму, тому задача їхньої інтеграції в рамках реляційної моделі потребує переробки всього рішення. Якщо ж конвертувати такі бази в модель RDF, то інтеграція зводиться до простого злиття RDF-графів і переписуванню запитів з SQL у SPARQL.

RDF-сховища найбільш придатні для задач, що потребують виявлення та аналізу великої кількості взаємозв'язків. До таких задач відносяться:

- обробка семантичних мереж (і інших графових структур), отриманих в результаті аналізу природномовних текстів;
- представлення й обробка даних з соціальних мереж (побудова портрета користувача, виявлення центрів поширення інформації у соціальних мережах тощо);
- обробка даних складних наукових експериментів.

Практично всі задачі, у яких кількість взаємозв'язків між сутностями перевищує кількість сутностей чи орієнтованих на аналіз взаємозв'язків, можуть розглядатися як кандидати на рішення засобами систем RDF.

Сучасні програмні засоби обробки неструктурованих даних

Існує велика кількість програмних засобів для обробки та керування НСД. Деякі з них використовують системи *керування корпоративним контентом* (CMS), що можуть підтримувати весь життєвий цикл його контента (Web-контент, документи тощо). Багато постачальників CMS масштабують свої рішення для обробки Big Data та орієнтовані на опрацювання великих обсягів НСД у реальному часі, використовуючи такі технології, як Hadoop, MapReduce і потокова передача.

Методи роботи з НСД іноді протиставляють технологіям BI, однак точніше говорити про їх взаємне доповнення [11]. Основний недолік BI пов’язують з їхньою недостатньою динамічністю та непристосованістю для обробки Big Data у режимі реального часу. Крім того, традиційні методи BI орієнтувалися на аналіз структурованої інформації. Інтеграція BI з технологіями обробки НСД називають *Embraced Enterprise Search and Retrieval* (ESR): в них реалізовано дві всеохоплюючі (Embraced) функції – корпоративний пошук (Enterprise Search) і здобування інформації з даних (Retrieval). ESR, крім доступу до нових типів даних, дозволяють здобувати більше корисної інформації також і зі звичайних структурованих даних.

Проблеми аналізу НСД загострилися через нові джерела таких даних – соціальні мережі, мобільні пристрої, реєстратори. Використання інформаційно-пошукових систем (ІПС), що традиційно застосовуються для пошуку в Web, ускладнюється великими обсягами та великою швидкістю накопичення НСД, що характерні для Big Data. Водночас як застосування для цього технологій корпоративного пошуку виявилося надто коштовним.

Середня довжина запитів до ІПС не перевищує двох-трьох слів, користувачі рідко застосовують логічні операції. У традиційних ІПС кожен запит виконується незалежно від попередніх, і пошукові машини дають ту саму відповідь будь-якому користувачу поза залежністю від передісторії його роботи з базою. Деякі компанії (наприклад, Google) використовують додаткову контекстну інформацію (метадані), що відноситься до предмета пошуку, та рейтинги сторінок. Але й такі системи не враховують особливості корпоративних даних, структурувати які все ж більш легко, ніж інформацію від довільних користувачів.

Задачі, які вирішують системи CMS, – оцінка причин відтоку клієнтів шляхом побудови профілів клієнтів, ана-

ліз відгуків та їх емоційного забарвлення, оцінка компаній у ЗМІ, внутрішні розслідування (пошук та захист від видалення документів, пов'язаних з певним інцидентом, в якому аналізуються НСД з різних корпоративних джерел – поштових серверів, корпоративних порталів, телефонних і відеоконференцій, та побудова взаємозв'язків між ними).

Засоби, що використовуються в CMS для аналізу НСД, порівнюють за наступними параметрами [12], значення яких наведено у табл. 1.

Таблиця 1. Параметри порівняння засобів CMS

Параметр	*Можливі значення*
Тип засобу	Засоби Text Mining Обробка контенту баз даних Інтеграція Text Mining та обробки контенту баз даних
Можливості	Аналіз ключових слів Статистичний аналіз Лінгвістичний аналіз
Джерела даних	Структуровані бібліографічні джерела даних Неструктуровані джерела даних Гібридні джерела даних
Результати	Списки документів Таблиці Графіки Карти

Для того, щоб визначити типові операції аналізу НСД в CMS, розглянемо кілька прикладів програмних продуктів, що широко застосовуються для такого аналізу.

Autonomy *IDOL* (Intelligent Data Operating Layer) [13] базується на обробці змісту (Meaning-Based Computing) текстів незалежно від форми їхнього представлення і форматів та забезпечує пошук понять (концептів) за пов'язаними з ними словами ПМ [14]. Для цього використовують різні підходи – пошук за ключовими словами, що враховує найпростіші закономірності (частоту повторень слів тощо), ранжирування (PageRank) на основі частоти звертань до того чи іншого документа, федеративний пошук (Federated Search), та концептуальний пошук (Conceptual Search) та мультимедійний пошук (Audio and Video Search), що сполучує власне пошук з розпізнаванням образів. Обробка змісту даних починається з їх класифікації та кластеризації. IDOL для розуміння змісту даних використовує метод байєсівського виведення (розрахунок імовірності події на основі статистики її здійснення в минулому) і теорію інформації Клода Шеннона разом із традиційними підходами до аналізу. Це дозволяє визначити категорії документів за допомогою статистичного аналізу слів, що зустрічаються в цих документах.

Endeca Latitude [15] – технологія Text Mining, що призначається для аналізу потоків сирої текстової інформації з різних джерел та фокусується на розкритті змісту даних на противагу традиційному аналізу. Вона містить *Latitude Information Integration Suite* – набір засобів для збору і попередньої обробки потоку сирих вхідних даних (структурованих, неструктурованих і квазіструктурованих), а також середовище для створення аналітичних застосувань *Latitude Studio* та гібридну пошуково-аналітичну СКБД з високою масштабованістю *MDEX Engine*.

Ця платформа забезпечує здобуття наступних п'яти типів інформаційних шаблонів, за допомогою яких користувачі задають режими створення цільових моделей пошуку інформації [16]:

1) оптимізація, що керується порівнянням (Analyze-Compare- Evaluate);

2) оптимізація, орієнтована на дослідження (Explore-Analyze-Evaluate);

3) стратегічний аналіз (Analyze-Comprehend-Evaluate);

4) стратегічний нагляд (Monitor-Analyze-Evaluate);

5) синтез, керований порівнянням (Analyze-Compare- Synthesize).

На вході Endeca Latitude працює *Latitude Information Integration Suite*, що складається з трьох основних компонентів:

- *Latitude Content Acquisition System* – система збору контента, що містить колекцію конекторів для виділення, очищення й інтеграції НСД з файлових систем, Web-сайтів тощо;
- *Latitude Data Integrator* – інтегратор, що виконує функції, аналогічні ETL (Extract, Transform, and Load – Витяг, Перетворення та Завантаження) у сховищах даних;
- Open *Interfaces and Connectors* – інтерфейси і конектори для отримання даних з Apache Hadoop та інших джерел.

MDEX Engine націлена на пошук і виявлення знань і є гібридом ІПС та аналітичної СКБД, що призначена для обробки даних, що швидко змінюються.

Принципова відмінність MDEX від традиційних СКБД полягає у наближенні записів, що зберігаються в ній, до реальностей навколишнього світу. Ці записи містять пари атрибутів "ключ/значення" (key/value). У формі атрибутів зберігаються ієрархічно організовані дані, наприклад елементи ієрархій XML, причому так, що користувач має можливість буквально угвинчуватися (drill-into) у набори даних, використовуючи для цього інструменти Latitude Studio. Таким чином MDEX дозволяє максимально позбутися процесів моделювання та працювати з даними у тому вигляді, як вони надійшли і зберігаються, – те, що називають «завантажив і пішов».

У MDEX реалізований фасетний пошук – пошук в інформаційних середовищах, побудованих за принципами *фасетної класифікації*.

Фасетна класифікація (класифікація двокрапкою, класифікація Ранганатана) – це сукупність кількох незалежних класифікацій, що здійснюються одночасно за різними базисами. В такій класифікації поняття представлені у вигляді перетину ряду ознак, а класифікаційні індекси синтезуються за допомогою комбінування фасетних ознак відповідно до фасетної формули [17].

Ця класифікація запропонована Шиалі Ранганатаном, відомого створенням "П'яти законів бібліотечної науки" (1931) [18], як варіант бібліотечно-бібліографічного підходу до багатоаспектної класифікації для звичайних паперових бібліотек і пізніше поширилися для комп'ютерних застосувань.

Це неієрархічна система організації інформації, у якій прості поняття розподілені у фасети – групи однорідних понять, пов'язані узагальненням за однією певною ознакою. Її структура є прямим відображенням системної характеристики класифікації, тобто базується на поділі об'єктів за кількома класифікаційними ознаками одночасно [19]. Фасетною ознакою може бути будь-яка класифікаційна ознака, яка використовується для угрупування понять у фасетні ряди, у результаті чого утворюються підкласи.

Особливість фасетної класифікації пов'язана з представленням фасетних ознак через їх послідовність, тобто результат класифікації залежить від впорядкування фасет (це визначає їх важливість для класифікації). Більш того, послідовність ознак у цій класифікації впливає на зміст поняття (наприклад, "процес: матеріал: устаткування: властивість"), яке визначає фасетна формула – індекс, що складається з послідовності фасетних ознак, розділених двокрапкою.

Такий підхід забезпечує багатоаспектний пошук інформації. У цій класифікації сполучаються індекси з різних таблиць у певних комбінаціях, що дозволяє отримати індекси для різноманітних предметів. Основна таблиця фасетної класифікації в кожній *предметній області* (ПрО) представлена набором таблиць, що будуються за класифікаційними ознаками (категоріями, фасетами) різного ступеня узагальнення – загальні (наприклад, "Властивості"), спільні для великих груп ПрО (наприклад, "Обладнання") та специфічні для окремих ПрО (наприклад, "Алгоритми сортування даних"). Таблиці таких категорій розробляються відповідно до специфіки кожної ПрО, а типові ознаки, характерні для більшості або всіх відділів фасетної класифікації, відображаються у

додаткових таблицях, а в особливій таблиці міститься визначення характеру зв'язків між поняттями ("Вплив", "Порівняння" тощо).

ClearForest (http://www.clearforest.com/Technology/) пропонує рішення Text Analytics, що містить платформу видобування тексту, аналітичну платформу та середовища розробки. Інструмент видобування тексту виконує генерування матриць спільного застосування термінів, кластеризації даних, видобутку термінів і тегування, тобто обирає відповідні терміни з неструктурованого тексту, наприклад, статей новин, Web-опитувань і документів HTML. Після структуризації ця інформація може бути використана в автономних аналітичних застосунках або в поєднанні зі структурованими даними, щоб забезпечити більш комплексний бізнес-інтелект. Терміни витягуються для подальшого аналізу і автоматично класифікуються в попередньо визначені категорії або таксономії.

Інструмент дозволяє візуалізувати взаємозв'язки між колекціями таксономій, щоб отримати інформацію, яка є актуальною, дієвою та додає цінності іншим інструментам Business Intelligence.

Однією з переваг ClearForest є перетворення НСД у структуровані дані за допомогою модуля Packaged Extraction Module. Наприклад, текст патентних документів перетворюється у структуровані таблиці з такими параметрами, як "проблеми" та "технологічні процеси".

Inxight (http://www.inxight.com/products/smartdiscovery_as/) – набір програмних рішень для аналізу ПМ дослідницького центру Xerox Palo Alto (PARC), що дозволяють розуміти документи настільки глибоко, щоб забезпечити їх індексацію, класифікацію та витяг всіх необхідних понять, сутностей та відношень. Програмне забезпечення ідентифікує більше 35 типів інформації в одному документі. Джерелами даних є текстові НСД, наприклад, новини, Web-сайти, внутрішні документи та повнотекстові патенти. Метадані і об'єкти можуть бути здобуті з попередньо оброблених документів. Результатом роботи Inxight є ієрархічна категоризація документів, тобто документи аналізуються на основі заздалегідь визначених категорій в ієрархіях.

Важливою особливістю Inxight є можливість одночасного пошуку в декількох онлайнових БД, відома як федеративний пошук ("fedederated search"). Inxight працює з 32 мовами і ідентифікує 27 типів об'єктів. Розробника Inxight стверджують, що лінгвістичні алгоритми, використані в цьому продукті, є найпотужнішими в даній галузі. Недоліком системи є потреба у значних витратах часу на аналіз тексту.

Платформа *Velocity Platform (http://vivisimo.com/html/velocity*) компанії Vivisimo складається з трьох пов'язаних програмних продуктів:

- *Search Engine* – багатофункціональна пошукова машина, агенти-краулери якої здатні аналізувати файли різних типів (HTML, TXT, RTF, Adobe Acrobat PDF, PostScript, MS Word, Excel, PowerPoint, WordPerfect, ZIP, GZIP, TAR Lotus Notes) та здобувати інформацію з реляційних СКБД різними мовами (всі європейські мови, арабська і китайська);
- *Clustering Mashine* – засіб кластеризації, що групує результати, отримані від Google, Autonomy, FAST і Ultraseek, а також тексти в різних форматах;
- *Content Integrator* – інтегратор, що забезпечує федерований пошук, що вміє працювати з метаданими і передавати результати до Clustering Engine.

Інші відомі програмні продукти, орієнтовані на аналіз текстових НСД, – це Goldfire Innovator (http://www.invention-machine.com/GoldfireInnovator.htm), OmniViz (http://www.biowisdom.com/solutions/), TEMIS (http://www.temis.com).

Сфера застосування засобів аналізу НСД

Призначення багатьох комерційних систем, що здійснюють аналіз текстових НСД, пов'язане з підтримкою зворотного зв'язку з клієнтами та аналізом емоційного інформаційного фону, що складається на-

вколо організації і її конкурентів [20]. Джерелами даних для них є ЗМІ, портали новин, соціальні мережі, аналітичні портали, внутрішні інформаційні ресурси компаній тощо. У цілому робота з НСД – це пошук і агрегація контента з різних джерел, витяг даних відповідно до заданих параметрів і їхній семантичний аналіз, а також надання підсумкових відомостей користувачу в зручному вигляді.

Наведемо ще кілька характерних прикладів таких систем:

- *First Rain* компанії First Rain – рішення для пошуку, збору й аналізу інформації тільки з Web-ресурсів (звітів компаній та аналітичних оглядів), яке класифікує знайдені відомості за стандартизованих набором тем і значущістю для клієнта;
- *Digimind* – рішення для пошуку структурованих і неструктурованих даних, з Web і соціальних мереж, що забезпечує класифікацію оброблених матеріалів та представлення підсумкових даних у вигляді, зручному для користувача;
- *InfoNgen* – набір рішень для пошуку, збору й аналізу НСД, що агрегують відомості з різних Web-джерел, електронної пошти та внутрішніх інформаційних ресурсів організації та категоризують їх відповідно до таксономії клієнта та дозволяють враховувати специфічні особливості кожного джерела;
- *Factiva* – набір інформаційно-аналітичних рішень, що дозволяє збирати мультимедійний контент з сайтів новин;
- *«Голос клієнта»* – рішення для аналізу структурованих і неструктурованих даних для обробки відгуків клієнтів з соціальних мереж, центрів роботи з клієнтами і CRM, форумів і блогів.

Семантичний аналіз НСД дозволяє визначити заголовок, резюме, зміст, дату публікації тощо, заданих користувачем елементів (наприклад, назв компаній, найменувань продуктів, послуг), відкинути непотрібні дані (рекламні оголошення, правові обмеження), розпізнати семантичну структуру тексту та семантичні залежності. У ході морфологічного і лексичного аналізу кожен текст розділяється на зв'язані між собою слова, що зіставляються з заздалегідь визначеними тегами. В процесі аналізу враховуються синоніми, можливі варіанти написання слів (іншими мовами або з типовими помилками), абревіатури.

Крім того, існує можливість визначення емоційної тональності тексту, що дозволяє оцінити відношення авторів документа до окремих інформаційних об’єктів, – позитивне чи негативне, а також задати цінність кожного позитивного і негативного висловлення в залежності від цілей користувача.

Постановка задачі

У зв’язку з тим, що велика частка інформаційних ресурсів – це неструктуровані текстові дані, виникає потреба у створенні засобів, що забезпечують здобуття з цих НСД тієї інформації, що необхідна користувачам для розв’язку їх поточних проблем. Використання традиційних засобів Text Mining може бути недостатньо ефективним для обробки Big Data, і це викликає необхідність інтелектуалізації засобів аналізу НСД. Основою такого аналізу може стати застосування фонових знань щодо предметної області, формалізованих за допомогою онтологій. Це викликає потребу у методах побудови спеціалізованих онтологій для задач користувачів та їх застосування для семантичної розмітки природномовних текстів. Пропонується використовувати для цього технології Wiki та їх семантичне розширення, а також створювати семантично розмічені Wiki-ресурси як основу для структурування довільних природномовних текстів.

Технологія Wiki як засіб структурування інформації

Під *Wiki-технологією* зазвичай розуміють таку технологію побудови Web-ресурсу, яка дає змогу відвідувачам брати участь у редагуванні його вмісту – виправляти помилки, додавати нові матеріали, не використовуючи спеціальні програми, явно вказувати зв’язки між окремими сторінками за допомогою гіперпосилань та

визначати категорії, до яких вони відносяться [21].

Формат Wiki-сторінок – це спрощена мова розмітки, що використовується для того, щоб виділити в тексті різні структурні й візуальні елементи або вказати на них. Важливою особливістю Wiki є те, що вносити структурування до текстових НСД за допомогою Wiki-розмітки може практично кожен користувач. На сьогодні існує велика кількість Wiki-двигунів та створених на їх основі розподілених інформаційних ресурсів різного обсягу та спрямованості. Найбільш великим та відомим з них є Вікіпедія.

Основними елементами Wiki-розмітки є гіперпосилання та категорії. Їх застосування дозволяє досить легко перетворювати НСД у частково структуровані дані. Крім того, аналіз структурування Wiki-ресурсів на рівні слів та понять дозволяє отримувати знання для структурування інших НСД.

Семантизація Wiki-ресурсів

Semantic MediaWiki (SMW) – це надбудова над інструментальним засобом побудови Wiki-сайту MediaWiki [22]. Переваги SMW – це обробка інформації на семантичному рівні, наявність засобів групового керування знаннями, відносно висока виразна потужність, надійна реалізація і зручний інтерфейс користувачів, наявність документації та спільнот користувачів [23]. Це дозволяє інтегрувати інформацію з різних Wiki-сторінок, здійснюючи пошук на рівні знань, та генерувати за Wiki-сторінками онтологічні структури, які можуть використовувати інші ІІС.

Крім категорій, в SMW для структурування інформації використовуються такі механізми, як *семантичні властивості*. Вони дозволяють семантично пов’язувати Wiki-сторінки як між собою, так і з різними даними. Кожна семантична властивість має тип, назву і значення, а також власну Wiki-сторінку в спеціальному просторі імен, яка дозволяє визначати її місце в ієрархії властивостей та документувати те, як цю властивість необхідно використовувати.

З точки зору онтологічного аналізу, кожна Wiki-сторінка являє собою онтологічний елемент, тобто елемент одного з RDF-класів – Thing, Class, ObjectProperty, DatatypeProperty, AnnotationProperty. Крім того, кожна стаття має власний URI, який дозволяє уникнути плутанини між поняттями і HTML- сторінками. Зазвичай, статті є екземплярами класів онтології OWL, категорії – класами, а відношення – об'єктними властивостями онтології.

Виходячи з цього, для будь-якої сторінки SMW за запитом може генерувати відповідний OWL/RDF-файл. Найпростіший спосіб отримати цей RDF – просто використати посилання "Переглянути як RDF" ("View as RDF"), що знаходиться в нижній частині кожної анотованої сторінки. Ця сторінка може виступати як кінцева точка (endpoint) для зовнішніх сервісів (зовнішньої точки доступу), які хочуть отримати доступ до семантичних даних SMW. На жаль, ця функція реалізована дуже невдало та підтримує надто мало опцій.

Оскільки SMW сумісна з моделлю знань OWL DL, то існує можливість використання в Wiki-ресурсах існуючих онтологій. Це можливо здійснити двома шляхами: імпорт онтології дозволяє створювати і модифікувати сторінки у Wiki для подання відношень, заданих в деякому існуючому OWL DL-документі; а повторне використання словника дозволяє користувачам відображати (задавати відповідності) Wiki-сторінки на елементи існуючих онтологій. Функція імпорту онтології для читання RDF-документів витягує RDF-твердження, які можуть бути представлені у Wiki.

Семантичні Wiki-ресурси можуть використовуватися як основа для автоматизованої генерації розподілених баз знань в форматі RDF. Експорт в OWL/RDF є засобом забезпечення зовнішнього повторного використання даних з Вікі, але тільки практичне застосування цієї функції може показати якість згенерованого RDF. З цією метою для видачі RDF, розробники системи використовували ряд інструментів Semantic Web.

Таким чином, наявність перевіреного та семантично розміченого Wiki-ресурсу дозволяє побудувати онтологію тієї ПрО, що цікавить користувача, яка може використовуватися в Text Mining для структурування НСД з цієї ПрО. Переваги використання моделі даних RDF вище наведені.

Основна проблема отримання фонових знань для Text Mining з Wiki-ресурсів пов'язана з тим, що сьогодні:

- преважна частка Wiki-ресурсів не семантизована;
- Wiki-ресурси здебільшого не є реферованими та авторськими, і тому наявність в них помилок (фактичних, структурних та змістовних) досить ймовірна;
- семантизовані Wiki-ресурси з високим рівнем довіри до контенту здебільшого високо спеціалізовані та орієнтовані на подання знань відносно вузьких ПрО (крім того, навіть в таких ресурсах зазвичай Wiki-онтологія, що лежить в основі їх семантичної розмітки, зазвичай не є доступною для зовнішніх користувачів);
- пошук та аналіз RDF та OWL із зовнішніх сховищ та репозиторіїв – досить складна задача, незважаючи на наявність спеціалізованих пошукових запитів, а знайдені таким чином онтології можуть не повністю відповідати поточним потребам користувача.

Крім вбудованих в Semantic MediaWiki засобів генерації RDF, існує багато більш спеціалізованих алгоритмів побудови онтологій на основі семантичної Wiki-розмітки [24]. Ці алгоритми дозволяють використовувати семантичний пошук та фонові знання щодо специфіки ПрО для формування корпусу Wiki-текстів, за якими створюється онтологія.

Це викликає потребу у розробці та вдосконаленні енциклопедичних онлайнових видань на базі семантичних Wiki-ресурсів. Саме до таких продуктів відноситься портальна версія Великої української енциклопедії е-ВУЕ. (http://vue.gov.ua), яка використовує вільне програмне забезпечення MediaWiki версії 1.29.1. та його семантичне розширення Semantic MediaWiki версії 2.5.5. (рис. 4). Це інноваційний проект із створення національної енциклопедії на основі сучасних засобів подання знань.

Рис. 4. Головна сторінка е-ВУЕ

Принциповими відмінностями е-ВУЕ від інших онлайнових довідників та енциклопедій (наприклад, від Вікіпедії) є:

- рецензованість – е-ВУЕ є науковим виданням з високою репутацією, яке подає експертні знання у вигляді авторських статей та надає перевірені сталі факти;
- наявність обробки інформації на рівні знань – пошук за семантичними властивостями;
- використання авторських статей.

Щоб використовувати семантизований Wiki-ресурс як розподілену базу знань (БЗ), створено Wiki-онтологію – модель знань цього ресурсу. Використання цієї моделі для семантичної розмітки забезпечує формування та програмної реалізації відповідного набору ієрархічно пов'язаних категорій, шаблонів типових інформаційних об'єктів, їх семантичних властивостей та запитів, що їх використовують. Наявність формальної моделі дозволить запобігти неоднозначної інтерпретації знань різними розробниками та користувачами ресурсу.

До пошуку подібних статей доцільно застосовувати принципи фасетної класифікації, тому що в е-ВУЕ для категоризації статей використовуються одночасно різні незалежні таксономії, такі як:

- галузі знань та їх підгалузі (рис. 5);

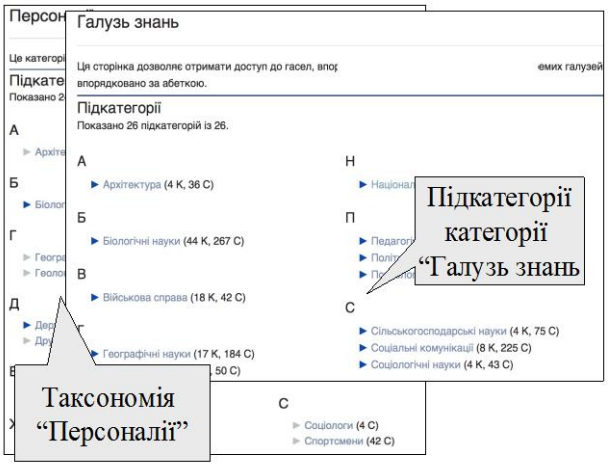


Рис. 5. Категорії e-BYE

- типові інформаційні об’єкти;
- наявність різних типів мультимедійного супроводу;
- таксономія географічних об’єктів;
- природа, цивілізація та персоналії;
- класифікація за авторами та модераторами.

Кожна стаття може використовувати один або кілька шаблонів для типових інформаційних об’єктів, що дозволяють задати значення семантичних властивостей сторінки та змістовно визначити її відношення з іншими сторінками енциклопедії.

Таким чином, *подібні* статті, – це статті, що віднесені до однакового або подібного набору категорії та мають подібні семантичні властивості, тобто аналіз близькості статей може оцінюватися через співставлення їх фасетних індексів.

Аналіз виразних можливостей розширених засобів подання та структурування інформації засобами технологічного середовища Semantic MediaWiki виявив, що, незважаючи на значно меншу, порівняно з онтологіями, їх виразну здатність, ці засоби дозволяють не тільки представляти класи та екземпляри онтологій, для яких існують однозначно визначені аналоги у Wiki [25] – категорії та Wiki-сторінки, але й представляти деякі більш складні знання. Запропоновано в цій роботі онтологічна модель Wiki-ресурсу дозволяє формально описувати такі характеристики семантичних властивостей різних типів, як припустимість неповних та множинних значень. Використання класів та екземплярів цієї онтології дозволяє генерувати Wiki-сторінки, на яких містяться результати виконання семантичних запитів, і за цими сторінками створювати онтології ПрО, що цікавлять користувачів. Застосування стандартів Semantic Web у Semantic MediaWiki забезпечує можливість використання цих онтологій у застосуваннях Text Mining без додаткової обробки.

Висновки

Проаналізувавши сучасні тенденції поширення неструктурованих текстових даних та засоби, що використовуються для їх аналізу, можна зробити висновки щодо високої актуальності цього напрямку та необхідність застосування до такої обробки інтелектуальних інформаційних систем. Big Data, значну частину яких складають саме неструктуровані тексти, потребують подальшого розвитку Text Mining та алгоритмів машинного навчання.

НСД, що складаються із природномовного тексту, у загальному випадку не мають попередньо визначеної моделі даних. Їх неоднозначність, гетерогенність та залежність від контексту значно ускладнюють класифікацію документів, ідентифікацію їх компонентів та автоматизоване здобуття з їх контенту знань, потрібних користувачеві, тоді як великі обсяги та динамічність таких даних не припускають ефективної ручної обробки.

Розглянуто засоби та методи структурування НСД, їх різноманітні програмні реалізації. Проаналізовано перспективи використання фонових знань для такого структурування. Обґрунтовано доцільність застосування для цього таких стандартів W3C, як RDF та OWL.

Використання семантичних Wiki-технологій для створення розподілених інформаційних ресурсів не тільки дозволяє досить легко додавати структурування до НСД, але й є джерелом фонових знань для аналізу довільних текстів відповідної предметної області. Запропоновані в роботі моделі та методи дозволяють вдосконалити цей процес.

Література

1. Grimes S. Unstructured Data and the 80 Percent Rule, 2008, Clarabridge, Bridgepoints. – http://breakthroughanalysis.com/2008/08/01 /unstructured-data-and-the-80-percent-rule/.
2. Неструктурированные данные в большой среде данных. – https://ru.howtodou.com/unstructured-data-in-big-data-environment.
3. Unstructured_data. – https://en.wikipedia.org/wiki/Unstructured_data.
4. Grimes S. A Brief History of Text Analytics. B Eye Network, 2016. – http://www.b-eye-network.com/view/6311.
5. Buneman P., Davidson S., Fernandez M., Suciu D. Adding structure to unstructured data. *International Conference on Database Theory*, 1997. P. 336–350.
6. Гладун А.Я., Рогушина Ю.В. Data Mining: пошук знань в даних. К.: ТОВ "ВД "АДЕФ-Україна", 2016. 452 с.
7. Feldman R., Sanger, J. The text mining handbook: advanced approaches in analyzing unstructured data. Cambridge university press, 2007. https://wtlab.um.ac.ir/images/e-library/text_mining/The%20Text%20Mining%20HandBook.pdf.
8. Шаховська Н. Особливості моделювання просторів даних, 2007. ena.lp.edu.ua/bitstream/ntb/35116/1/24_139-148.pdf.
9. Sadalage P., Fowler M. NoSQL Distilled. Pearson Education, 2012. 192 p.
10. Головков В., Портнов А., Чернов В. RDF – инструмент для неструктурированных данных. *Открытые системы. СУБД.* https://www.osp.ru/os/2012/09/13032513/.
11. Черняк Л. Аналитика неструктурированных данных. *Открытые системы*. 2012, № 06.
12. Yang Y., Akers L., Klose T., Yan, C. B. Text mining and visualization tools–impressions of emerging capabilities. World Patent Information. 2008. 30(4). P. 280–293. https://www.scss.tcd.ie/Khurshid.Ahmad/Research/High_Frequency_Trading/2008_Yanget al_TextMiningVis_ WorldPatent.pdf.
13. Autonomy IDOL. http://www.autonomy.com/content/Products/products-idol-server/index.en.html.
14. Lyte V., Jones S., Ananiadou S., Kerr L. UK institutional repository search: innovation and discovery, 2009. http://www.ariadne.ac.uk/issue/61/ lyte-et-al/.
15. Russell-Rose T., Lamantia J., Burrell M. A Taxonomy of Enterprise Search. *EuroHCIR*, 2011. P. 15–18. https://www.researchgate.net/profile/Joe_Lamantia/publication/235971352_A_Taxonomy_of_Enterprise_Search_and_Discovery/links/00b7d515063de775c8000000.pdf.
16. Lamantia J. 10 Information Retrieval Patterns, 2006. http://www.joelamantia.com/ information-architecture/10-information-retrieval-patterns.
17. Фасетна класифікація. http://uk.wikipedia.org/wiki/ Фасетна_класифікація.
18. Noruzi A. Application of Ranganathan's Laws to the Web. http://www.webology.org/2004/v1n2/a8.html.
19. Сербин О.О. Особенности фасетной классификации документов в условиях современной трансформации содержания науки о книге. Книжная культура в контексте международных контактов: Материалы III Международной научной конференции, Минск: ЦНБ НАН Беларуси; М.: ФГБУН НИЦ «Наука» РАН, 2015. С. 457–462. http://eprints.rclis.org/25289/1/serbin.pdf.
20. Оганесян А. Неструктурированные данные 2.0. *Открытые системы. СУБД*. 2012. № 04. https://www.osp.ru/os/2012/04/13015772/.
21. Wagner C. Wiki: A technology for conversational knowledge management and group collaboration. *The Communications of the Association for Information Systems*. 2004. Vol. 13(1). P. 264–289. http://aisel.aisnet.org/cgi/viewcontent.cgi? article= 3238& context= cais.
22. MediaWiki. https://www.mediawiki.org/wiki/MediaWiki.
23. 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.
24. Rogushina J. Processing of Wiki Resource Semantics on Base of Ontological Analysis. Proc.of VIII International scientific conference «Open Semantic Technologies for Intelligent Systems» OSTIS-2018, Minsk, 2018. P. 159–162. https://libeldoc.bsuir.by/bitstream/123456789/30389/1/Rogushina_Processing.PDF.
25. Rogushina J. Analysis of Automated Matching of the Semantic Wiki Resources with Elements of Domain Ontologies.

International Journal of Mathematical Sciences and Computing (*IJMSC*). 2017. Vol. 3. N 3. P. 50–58. http://www.mecs-press.org/ijmsc/ijmsc-v3-n3/IJMSC-V3-N3-5.pdf.

References

1. Grimes S. Unstructured Data and the 80 Percent Rule, 2008, Clarabridge, Bridgepoints. http://breakthroughanalysis.com/2008/08/01 /unstructured-data-and-the-80-percent-rule/.
2. Unstructured data in big data environment. https://ru.howtodou.com/unstructured-data-in-big-data-environment.
3. Unstructured_data. https://en.wikipedia.org/wiki/Unstructured_data.
4. Grimes S. A Brief History of Text Analytics. B Eye Network, 2016. http://www.b-eye-network.com/view/6311.
5. Buneman P., Davidson S., Fernandez M., Suciu D. Adding structure to unstructured data. // International Conference on Database Theory, 1997. P. 336–350.
6. Gladun A.Ya., Rogushina Y.V. Data Mining: Finding Knowledge in Data. K .: ADEF-Ukraine Ltd., 2016. 452 p. [in Ukrainian]
7. Feldman R., Sanger, J. The text mining handbook: advanced approaches in analyzing unstructured data. Cambridge university press, 2007. https://wtlab.um.ac.ir/images/e-library/text_mining/The%20Text%20Mining%20HandBook.pdf.
8. Shakhovska N. Features of modeling of data spaces, 2007. ena.lp.edu.ua/bitstream/ntb/35116/1/24_139-148.pdf. [in Ukrainian]
9. Sadalage P., Fowler M. NoSQL Distilled. Pearson Education, 2012. 192 p.
10. Golovkov V., Portnov A., Chernov V. RDF as a tool for unstructured data // Open Systems. https://www.osp.ru/os/2012/09/13032513/. [in Russian]
11. Chernyak L. Analytics of unstructured data. Open systems, 2012, № 06. [in Russian]
12. Yang Y., Akers L., Klose T., Yan, C. B. Text mining and visualization tools–impressions of emerging capabilities. World Patent Information, 30(4), 2008. P. 280–293. https://www.scss.tcd.ie/Khurshid.Ahmad/Research/High_Frequency_Trading/2008_Yangetal_TextMiningVis_ WorldPatent.pdf.
13. Autonomy IDOL. http://www.autonomy.com/content/Products/products-idol-server/index.en.html.
14. Lyte V., Jones S., Ananiadou S., Kerr L. UK institutional repository search: innovation and discovery, 2009. http://www.ariadne.ac.uk/issue/61/ lyte-et-al/.
15. Russell-Rose T., Lamantia J., Burrell M. A Taxonomy of Enterprise Search // EuroHCIR, 2011. P. 15–18. https://www.researchgate.net/profile/Joe_Lamantia/publication/235971352_A_Taxonomy_of_Enterprise_Search_and_Discovery/links/00b7d515063de775c8000000.pdf.
16. Lamantia J. 10 Information Retrieval Patterns, 2006. http://www.joelamantia.com/information-architecture/10-information-retrieval-patterns.
17. Faceted classification. http://uk.wikipedia.org/wiki/ Фасетна_класифікація. [in Russian]
18. Noruzi A. Application of Ranganathan's Laws to the Web. http://www.webology.org/2004/v1n2/a8.html.
19. Serbin O.O. Features of the faceted classification of documents under the modern transformation of the book science content // Book culture in the context of international contacts: Proc| of the III International Scientific Conference, Minsk: Central Scientific Library of the National Academy of Sciences of Belarus, 2015. P. 457–462. http://eprints.rclis.org/25289/1/serbin.pdf. faceted classification. [in Russian]
20. Oganesyan A. Unstructured Data 2.0 // Open Systems. N 04, 2012. https://www.osp.ru/os/2012/04/13015772/. [in Russian]
21. Wagner C. Wiki: A technology for conversational knowledge management and group collaboration // The Communications of the Association for Information Systems. 2004. Vol. 13(1). P. 264–289. http://aisel.aisnet.org/cgi/viewcontent.cgi? article= 3238& context= cais.
22. MediaWiki. https://www.mediawiki.org/wiki/MediaWiki.
23. Rogushina Y.V., Priyma S.M, Strokan O.V. Creating and use of the Semantic Wiki resources: tutorial. Melitopol, FOP Odnorog T.V., 2017. 169 p. [in Ukrainian]
24. Rogushina J. Processing of Wiki Resource Semantics on Base of Ontological Analysis. Proc.of VIII International scientific conference «Open Semantic Technologies for Intelligent Systems» OSTIS-2018, Minsk,

2018. P. 159–162. https://libeldoc.bsuir.by/bitstream/123456789/30389/1/Rogushina_Processing.PDF.
25. Rogushina J. Analysis of Automated Matching of the Semantic Wiki Resources with Elements of Domain Ontologies. International Journal of Mathematical Sciences and Computing (IJMSC). 2017. Vol. 3. N 3. P. 50–58. http://www.mecs-press.org/ijmsc/ijmsc-v3-n3/IJMSC-V3-N3-5.pdf.

Одержано 06.02.2019

Про автора:

Рогушина Юлія Віталіївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 140.
Кількість наукових публікацій в зарубіжних виданнях – 30.
Індекс Хірша – 3.
http://orcid.org/0000-0001-7958-2557.

Місце роботи автора:

Інститут програмних систем
НАН України,
03181, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: 066 550 1999.
E-mail: ladamandraka2010@gmail.com

УДК 004.94+004.896 https://doi.org/10.15407/pp2019.01.078

А.Л. Яловець

МЕТОДИ РОЗПІЗНАВАННЯ АГЕНТОМ НЕВІДОМОГО НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Викладено методи, що використовуються агентом для розпізнавання невідомого навколишнього середовища, в тому числі: метод визначення відстані до найближчого видимого об’єкта та координат перетину з цим об’єктом променя (як модельованого напряму зору агента), направленого від агента; метод динамічної зміни градації кута променя, який направляється агентом у навколишнє середовище; метод побудови множини точок, що належать видимим з точки поточного розташування агента оточуючим об’єктам навколишнього середовища; метод узагальнення агентом множини точок з побудовою фрагментів семантичної мапи невідомого навколишнього середовища; методи розпізнавання кутів приміщень у навколишньому середовищі.

Ключові слова: агент, невідоме навколишнє середовище, семантична мапа.

Вступ

В роботі [1] виконано постановку задачі розпізнавання невідомого навколишнього середовища, яка, зокрема, включає до свого складу задачу побудови агентом мапи цього середовища. Слід зауважити, що виконана постановка задачі за своїм змістом передбачала побудову «семантичної мапи» навколишнього середовища, під якою розуміється [2] мапа, яка додатково до просторової інформації про навколишнє середовище також містить семантичні властивості, зіставлені сутностям відомих класів, розташованим на мапі. Такі семантичні властивості дозволяють розрізнити, зокрема, як типи приміщень (наприклад, кімнату від коридору), так і об’єкти, що розташовані у приміщеннях (наприклад, столи від стільців). Виходячи з цього, відповідно до переліку задач досліджень, сформульованих в [1], в даній статті викладено методи розпізнавання агентом невідомого навколишнього середовища, на основі яких агент будує семантичну мапу цього середовища. До таких методів, зокрема, належать: метод визначення агентом відстані до найближчого до нього видимого об’єкта та координат перетину з цим об’єктом променя (як модельованого напряму зору агента), направленого від агента; метод динамічної зміни агентом градації кута променя, який ним направляється в навколишнє середовище; метод побудови агентом множини точок, що належать видимим з точки поточного розташування агента оточуючим об’єктам навколишнього середовища, отриманих в результаті направлення ним в навколишнє середовище променів з визначеним змінним кутом; метод узагальнення агентом множини точок з побудовою фрагментів семантичної мапи невідомого навколишнього середовища; методи розпізнавання кутів приміщень у навколишньому середовищі.

1. Метод визначення відстані до найближчого видимого об’єкта

Припустимо, що рухомому агенту необхідно розпізнати видимі об’єкти, що його оточують. Для цього він у кожний момент часу свого руху сканує оточуюче його середовище, послідовно направляючи відносно себе промені з певним кроком кутів. Розглянемо довільну ситуацію, коли агент у деякий момент часу розташований у точці A і з метою розпізнавання оточуючих його об’єктів сформував промінь AB для визначення відстані (довжини прямої AE) до найближчого об’єкта та координат x_E, y_E точки перетину променя з цим об’єктом точки E (рис. 1).

Також припустимо, що на даному рисунку нам відомі тільки координати трьох точок (A, C, D) та кут α_{AB} напряму променя щодо поточного розташування агента (точки A) в Декартовій системі координат та координата x_B «глибини» сканування навколишнього середовища («глибина» сканування визначається на початковому етапі розпізнавання), де C, D –

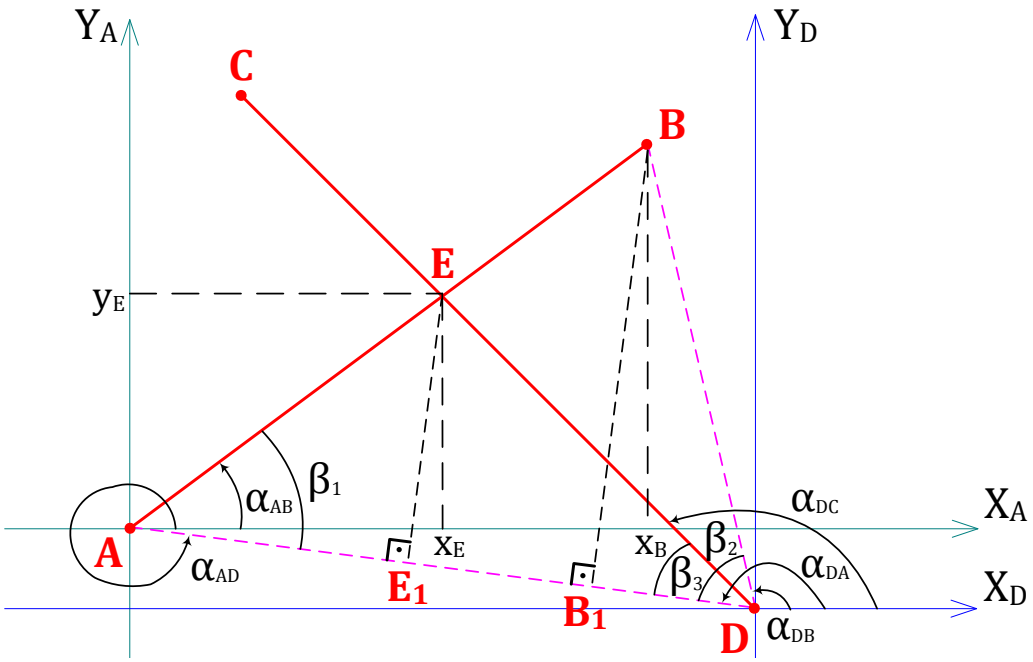


Рис. 1. Схема розташування агента щодо об’єкта, що потребує розпізнавання

координати меж деякого лінійного об’єкта, що потребує розпізнавання (які зберігаються у БД опису навколишнього середовища).

Кінцева мета процесу розпізнавання навколишнього середовища агентом полягає у формуванні ним власної БД опису навколишнього середовища.

Виконаємо необхідні добудови. Прокладемо осі координат щодо точок A і D (вважаємо, що агент функціонує в декартовій системі координат).

З’єднаємо точки A, D та B, D прямими, а з точок E і B проведемо перпендикуляри на пряму AD. Очевидно, що утворені при цьому трикутники AEE_1 та ABB_1 є подібними і для них справедливе співвідношення:

$$\frac{AB}{AE} = \frac{BB_1}{EE_1},$$

(так як $\sin\beta_1 = \frac{EE_1}{AE}$ та $\sin\beta_1 = \frac{BB_1}{AB}$).

Звідси маємо шукану довжину AE:

$$AE = \frac{AB \cdot EE_1}{BB_1}. \qquad (1)$$

Знаючи AE, легко визначити шукані координати x_E та y_E ($x_E = AE \cdot \cos\alpha_{AB}$; $y_E = AE \cdot \sin\alpha_{AB}$).

У виразі (1) нам невідомі EE_1 та BB_1.

Для визначення BB_1 розглянемо прямокутний трикутник DBB_1, в якому нам відомі гіпотенуза BD та кут β_2 (де $\beta_2 = \alpha_{DA} - \alpha_{DB}$). Звідси $BB_1 = BD \cdot \sin\beta_2$.

Для визначення EE_1 розглянемо два прямокутних трикутника AEE_1 та DEE_1, у яких є загальний катет EE_1. Кути β_1 ($\beta_1 = \alpha_{AB} + \alpha_{AD}$) та β_3 ($\beta_3 = \alpha_{DA} - \alpha_{DC}$) відомі.

Враховуючи, що $AE_1 + E_1D = AD$, та приймаючи $AE_1 = x$, маємо: $x \cdot tg\beta_1 = (AD - x) \cdot tg\beta_3$.

Звідси $x = AD \cdot \frac{tg\beta_3}{tg\beta_1 + tg\beta_3}$.

Враховуючи, що $EE_1 = AE_1 \cdot tg\beta_1$, маємо:

$$EE_1 = AD \cdot \frac{tg\beta_1 \cdot tg\beta_3}{tg\beta_1 + tg\beta_3},$$

або, після перетворень:

$$EE_1 = AD \cdot \frac{\sin\beta_1 \cdot \sin\beta_3}{\sin(\beta_1 + \beta_3)}.$$

Перетворюючи вираз (1), остаточно маємо:

$$AE = AB \cdot \frac{AD}{BD} \cdot \frac{\sin\beta_1 \cdot \sin\beta_3}{\sin\beta_2 \cdot \sin(\beta_1 + \beta_3)}. \qquad (2)$$

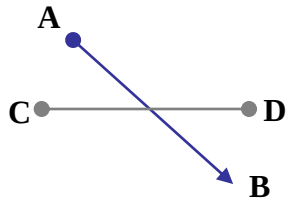
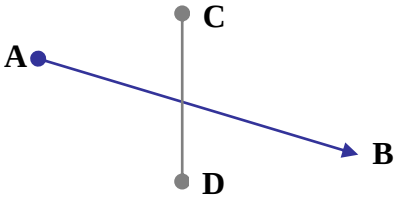
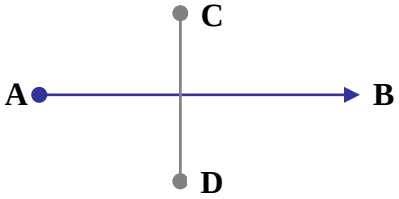
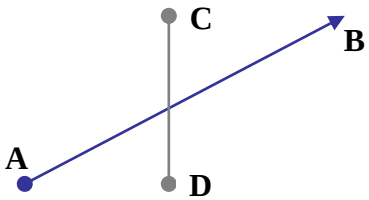
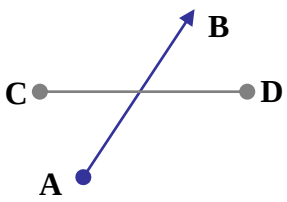
Тоді

$$x_E = AB \cdot \frac{AD}{BD} \cdot \frac{\cos\alpha_{AB} \cdot \sin\beta_1 \cdot \sin\beta_3}{\sin\beta_2 \cdot \sin(\beta_1 + \beta_3)},$$

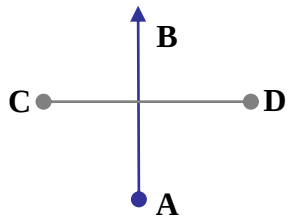
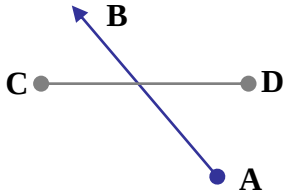
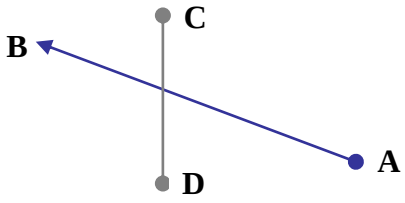
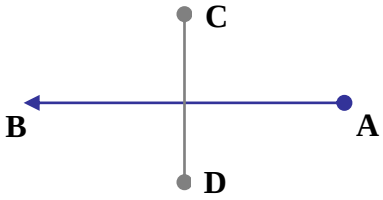
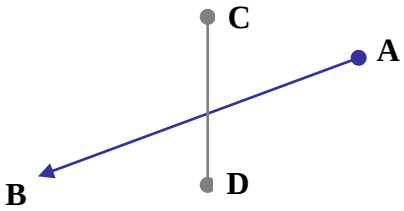
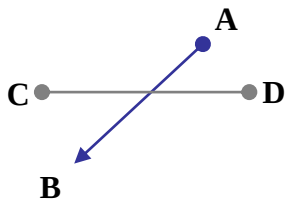
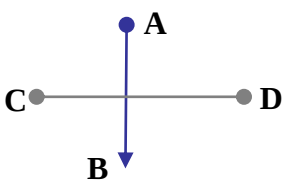
$$y_E = AB \cdot \frac{AD}{BD} \cdot \frac{\sin\alpha_{AB} \cdot \sin\beta_1 \cdot \sin\beta_3}{\sin\beta_2 \cdot \sin(\beta_1 + \beta_3)}. \qquad (3)$$

Оскільки агент може рухатись у різних напрямах і об’єкти, що потребують розпізнавання, можуть розташовуватись з різних сторін від нього, то формули визначення кутів $\beta_1, \beta_2, \beta_3$ будуть різними у кожному окремому випадку (таблиця).

Таблиця. Класифікація ситуацій розпізнавання агентом оточуючих об’єктів

№ сит.	Напрям променя / розташування стіни	Примітки	
1	2	3	4
1		Промінь у IV октанті; стіна горизонтальна	$\beta_1 = \alpha_{AD} - \alpha_{AB}$ $\beta_2 = \alpha_{DB} - \alpha_{DA}$ $\beta_3 = 180^\circ - \alpha_{DA}$
2		Промінь у IV октанті; стіна вертикальна	$\beta_1 = \alpha_{AB} - \alpha_{AD}$ $\beta_2 = \alpha_{DA} - \alpha_{DB}$ $\beta_3 = \alpha_{DA} - \alpha_{DC}$
3		Промінь на стику I і IV октантів; стіна вертикальна	$\beta_1 = 360^\circ - \alpha_{AD}$ $\beta_2 = \alpha_{DA} - \alpha_{DB}$ $\beta_3 = \alpha_{DA} - 90^\circ$
4		Промінь у I октанті; стіна вертикальна	$\beta_1 = (360^\circ - \alpha_{AD}) + \alpha_{AB}$ $\beta_2 = \alpha_{DA} - \alpha_{DB}$ $\beta_3 = \alpha_{DA} - 90^\circ$
5		Промінь у I октанті; стіна горизонтальна	$\beta_1 = \alpha_{AB} - \alpha_{AD}$ $\beta_2 = \alpha_{DA} - \alpha_{DB}$ $\beta_3 = \alpha_{DA} - 180^\circ$

Закінчення таблиці

1	2	3	4
6		Промінь на стику I і II октантів; стіна горизонтальна	$\beta_1 = 90^\circ - \alpha_{AD}$ $\beta_2 = \alpha_{DA} - \alpha_{DB}$ $\beta_3 = \alpha_{DA} - 180^\circ$
7		Промінь у II октанті; стіна горизонтальна	$\beta_1 = \alpha_{AB} - \alpha_{AD}$ $\beta_2 = \alpha_{DA} - \alpha_{DB}$ $\beta_3 = \alpha_{DA} - 180^\circ$
8		Промінь у II октанті; стіна вертикальна	$\beta_1 = \alpha_{AD} - \alpha_{AB}$ $\beta_2 = \alpha_{DB} - \alpha_{DA}$ $\beta_3 = 90^\circ - \alpha_{DA}$
9		Промінь на стику II і III октантів; стіна вертикальна	$\beta_1 = \alpha_{AD} - 180^\circ$ $\beta_2 = \alpha_{DB} - \alpha_{DA}$ $\beta_3 = 90^\circ - \alpha_{DA}$
10		Промінь у III октанті; стіна вертикальна	$\beta_1 = \alpha_{AD} - \alpha_{AB}$ $\beta_2 = \alpha_{DB} - \alpha_{DA}$ $\beta_3 = 90^\circ - \alpha_{DA}$
11		Промінь у III октанті; стіна горизонтальна	$\beta_1 = \alpha_{AD} - \alpha_{AB}$ $\beta_2 = \alpha_{DB} - \alpha_{DA}$ $\beta_3 = 180^\circ - \alpha_{DA}$
12		Промінь на стику III і IV октантів; стіна горизонтальна	$\beta_1 = \alpha_{AD} - 270^\circ$ $\beta_2 = \alpha_{DB} - \alpha_{DA}$ $\beta_3 = 180^\circ - \alpha_{DA}$

Отже, в результаті виконання класифікації виявлено дванадцять унікальних ситуацій розпізнавання агентом об'єктів спостережуваного навколишнього середовища.

2. Метод динамічної зміни агентом градації кута променя, який ним направляється в навколишнє середовище

Метод ґрунтується на поєднанні ідей, що лежать в основі кліткової декомпозиції та обробки графу огляду [1]. У випадку, коли градація кута променя є постійною, агент може втрачати інформацію [1]. Як наслідок, для забезпечення потрібної якості формування інформації агентом про навколишнє середовище необхідно, щоб ця градація динамічно змінювалась у залежності від відстані до об'єкта, що спостерігається агентом. При цьому, як зауважено в [1], за постановкою задач досліджень навколишнім середовищем для агента є будівлі ортогональної конфігурації в 2D проекції з довільною кількістю стін та отворів. Відомо, що для таких будівель властива певна модульність геометричних розмірів їх складових (як стін, так і отворів). Для вирішення задачі динамічної зміни агентом градації кута променя доцільно ввести поняття модуля (де розміри стін та отворів будуть кратними його значенню), який буде використовуватись для розбиття простору, що відповідає навколишньому середовищу агента, на клітинки, де розмір кожної її сторони буде відповідати значенню модуля (очевидно, що цей підхід подібний за своєю ідеєю сутності підходу кліткової декомпозиції). Отже, на рис. 2 показано, що агенту необхідно визначити градацію кута $\Delta\alpha$ як $\arctan\left(\frac{H}{X}\right)$, де H – розмір сторони клітинки (значення модуля), X – максимальне значення по осі X відстані до найближчого об'єкта, видимого з точки поточного розташування агента (в загальному випадку – це максимальне значення різниці між координатами X (або Y) точки перетину променя з таким об'єктом та координатами розташування агента).

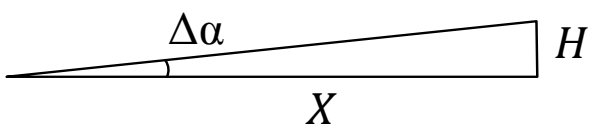


Рис. 2. Приклад визначення значення градації кута огляду агента

3. Метод побудови агентом множини точок, що належать видимим агенту оточуючим об'єктам навколишнього середовища

Метод ґрунтується на тому, що агент *послідовно* направляє промені проти часової стрілки, кожного разу змінюючи кут на обчислюване значення $\Delta\alpha$, та визначаючи (за допомогою виразів (3)) координати точок перетину променів з найближчими видимими об'єктами. Як наслідок, у агента формується множина координат таких точок, що є упорядкованою (як і у випадку обробки графу огляду), і тому дозволяє стверджувати, що подальший послідовний аналіз таких точок дозволить згенерувати семантичну мапу навколишнього середовища. На рис. 3 показано процес формування агентом множини точок, що належать видимим агенту об'єктам навколишнього середовища, виконуваний засобами мультиагентної системи (МАС) САГА, призначеної для моделювання поведінки агентів (у тому числі побудови семантичної мапи) у невідомому навколишньому середовищі. Як видно з цього рисунку, градація кута огляду агента є змінною.

4. Метод узагальнення агентом множини точок з побудовою фрагментів семантичної мапи невідомого навколишнього середовища

Як зазначено у вступі, під «семантичною мапою» навколишнього середовища розуміється [2] мапа, яка додатково до просторової інформації про навколишнє середовище також містить семантичні властивості, зіставлені сутностям відомих класів, розташованим на мапі. Будемо вважати базовими сутностями, що

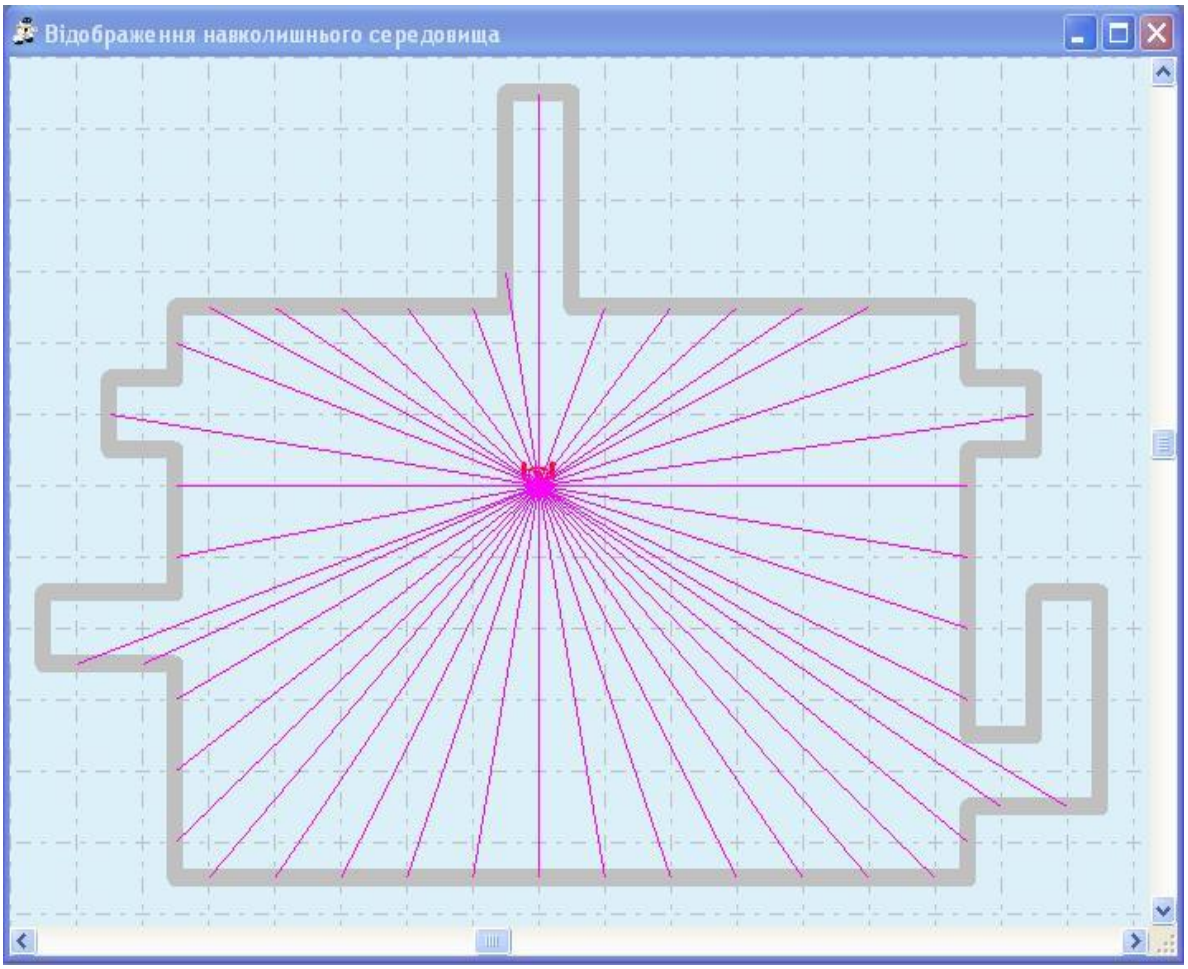


Рис. 3. Процес формування агентом множини точок

розташовані на мапі та необхідні для розпізнавання більш складних сутностей, стіни та отвори, які належать приміщенням, що розпізнаються. Тобто на початковому етапі процесу побудови семантичної мапи агенту необхідно сформувати фрагменти цієї мапи у вигляді стін та отворів, а далі, шляхом узагальнення цієї інформації, отримати більш загальну семантичну інформацію про навколишнє середовище (типи приміщень тощо).

На даному етапі будемо вважати, що процес створення семантичної мапи SM_a агентом a може бути поданий у вигляді кортежу (4):

$$SM_a = < P, R > . \qquad (4)$$

Тут P – множина точок, побудована агентом (див. п. 3 статті); R – правила перетворення агентом множини P у фрагменти семантичної мапи (стіни та отвори).

Правила R можуть бути структуровані на три групи: правила розпізнавання «горизонтальних» стін (у яких збігається координата Y в декартовій системі координат); правила розпізнавання «вертикальних» стін (у яких, відповідно, збігається координата X); правила розпізнавання та обробки ситуацій зміни типу стіни («горизонтальної» або «вертикальної»), що «спостерігається» агентом, або геометрії стіни (ці правила відповідають ситуаціям розпізнавання різного роду кутів приміщень).

Фрагмент лістингу формалізації правил розпізнавання «горизонтальних» стін за допомогою предикатів мови програмування Prolog показано на рис. 4.

```
f_wall(pnt(X1,Y1),pnt(X2,Y2),S,S1):-
% предикат f_wall перевіряє дві послідовні точки pnt (з множини точок) і, якщо координати Y  у них збігаються, формує дані щодо
% «горизонтальної» стіни; S – стан стіни, що розпізнано останньою; S1 – стан стіни, що розпізнається в даний момент часу
        Y1 = Y2,!, prov_SG(pnt(X1,Y1),pnt(X2,Y2),S,S1).

prov_SG(pnt(X1,Y),pnt(X2,Y),S,S1):-
% перший клоз предиката prov_SG аналізує випадок, коли відбувається зміна стану стіни (з невизначеної до горизонтальної)
        S=" ",!, S1="gor", n_w(N), N1=N+1,
% перевизначення номеру стіни, що оброблюється
        retract(n_w(_),envir_w),!, assert(n_w(N1),envir_w),
        f_w3(N1,Y,X1,X2).
prov_SG(pnt(X1,Y),pnt(X2,Y),S,S1):-
% другий клоз предиката prov_SG аналізує випадок, коли стан стіни не змінюється
        S="gor",!, S1=S, f_w4(Y,X1,X2).
prov_SG(PNT1,PNT2,S," "):-
% третій клоз предиката prov_SG аналізує випадок, коли відбувається зміна стану стіни (з горизонтальної до невизначеної)
        f_w5(S,PNT1,PNT2),!.

f_w3(N,Y,X1,X2):-
        X2>X1,!, X11 = X1 - step/2.0, X21 = X2 + step/2.0,  % step – значення модулю
% збереження предикату БД work_w як робочого стану стіни, що оброблюється
        assert(work_w(N,"Горизонтальна стіна",pnt(X11,Y),pnt(X21,Y)),envir_w).
f_w3(N,Y,X1,X2):-
        X11 = X1 + step/2.0,  X21 = X2 - step/2.0,
        assert(work_w(N," Горизонтальна стіна ",pnt(X11,Y),pnt(X21,Y)),envir_w).

f_w4(Y,X1,X2):-
        abs(X1-X2) = step,!, % стіна продовжується (без отвору)
        f_w41(Y,X1,X2).
f_w4(Y,X1,X2):-      % у межах стіни виявлено отвір
        X2>X1,!, X11 = X1 + step/2.0, work_w(N,S,pnt(X0,Y),_),
% вилучення з БД предикату work_w робочого стану стіни та додавання предикату БД wall_r як опису розпізнаної стіни
        retract(work_w(_,_,_,_),envir_w),!, assert(wall_r(N,S,pnt(X0,Y),pnt(X11,Y)),envir_r),
        X21 = X2 - step/2.0, n_p(NP), NP1 = NP +1,
        retract(n_p(_),envir_w),!, assert(n_p(NP1),envir_w),
% додавання предикату БД pr_r як опису розпізнаного отвору
        assert(pr_r(NP1,pnt(X11,Y),pnt(X21,Y)),envir_r), X22 = X21 + step, N1 = N+1,
        retract(n_w(_),envir_w),!,
        assert(n_w(N1),envir_w),
        assert(work_w(N1,S,pnt(X21,Y),pnt(X22,Y)),envir_w).
```

Рис. 4. Фрагмент лістингу з формалізації розпізнавання агентом «горизонтальних» стін

На рис. 5 показано приклад розпізнавання приміщення агентом з формуванням ним фрагментів семантичної мапи (з окремої точки розташування агента).

На рис. 6 показано фрагмент БД опису навколишнього середовища, сформованого в середовищі МАС САГА вручну шляхом побудови приміщень з окремих семантично не інтерпретованих ліній (відповідає рис. 3). На рис. 7 показано БД опису цього ж навколишнього середовища як фрагментів семантичної мапи, сформованої агентом у середовищі МАС САГА (відповідає стінам чорного кольору на рис. 5).

Слід підкреслити, що на відміну від БД стін, яка може створюватись з фрагментів ліній і не відповідати за змістом фактичній структурі окремої стіни, БД фрагменту семантичної мапи, як випливає з рис. 7, містить цілісне подання стін як семантично завершене.

В процесі руху агент постійно уточнює і добудовує семантичну мапу (рис. 8, зліва направо).

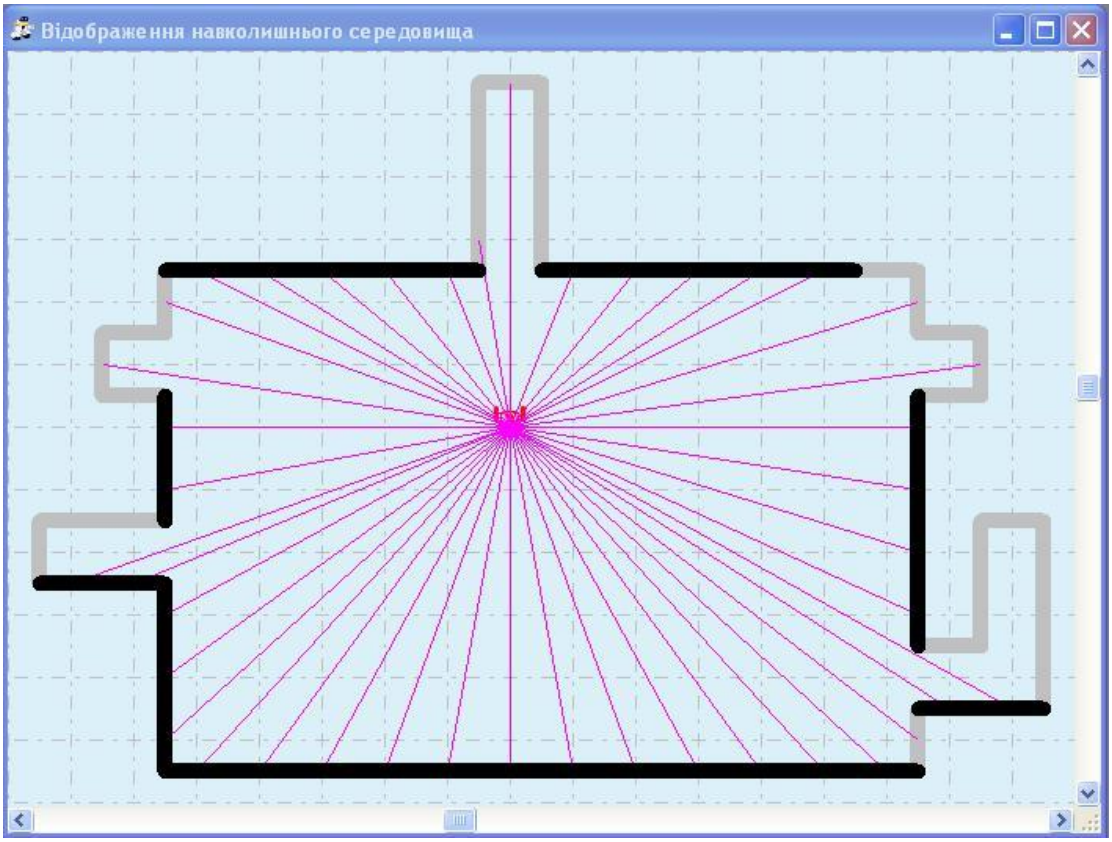


Рис. 5. Процес побудови фрагментів семантичної мапи

```
wall(1,pnt(8360,5080),pnt(8360,5200))
wall(2,pnt(8360,5200),pnt(8160,5200))
wall(4,pnt(8160,5400),pnt(8160,5520))
wall(5,pnt(8160,5520),pnt(8640,5520))
wall(6,pnt(8640,5520),pnt(8640,5480))
wall(7,pnt(8640,5440),pnt(8640,5280))
wall(8,pnt(8640,5240),pnt(8640,5200))
wall(11,pnt(8640,5200),pnt(8400,5200))
wall(12,pnt(8400,5200),pnt(8400,5080))
wall(13,pnt(8400,5080),pnt(8360,5080))
……
wall(27,pnt(8160,5280),pnt(8120,5280))
wall(28,pnt(8120,5280),pnt(8120,5240))
wall(29,pnt(8120,5240),pnt(8160,5240))
```

Рис. 6. Фрагмент БД стін

```
wall_r(1,"Горизонтальна стіна",pnt(8160,5520),pnt(8640,5520))
wall_r(2,"Горизонтальна стіна",pnt(8640,5480),pnt(8720,5480))
wall_r(3,"Вертикальна стіна",pnt(8640,5440),pnt(8640,5280))
wall_r(4,"Горизонтальна стіна",pnt(8600,5200),pnt(8400,5200))
wall_r(5,"Горизонтальна стіна",pnt(8360,5200),pnt(8160,5200))
wall_r(6,"Вертикальна стіна",pnt(8160,5280),pnt(8160,5360))
wall_r(7,"Горизонтальна стіна",pnt(8080,5400),pnt(8160,5400))
wall_r(8,"Вертикальна стіна",pnt(8160,5400),pnt(8160,5520))
```

Рис. 7. БД фрагменту семантичної мапи

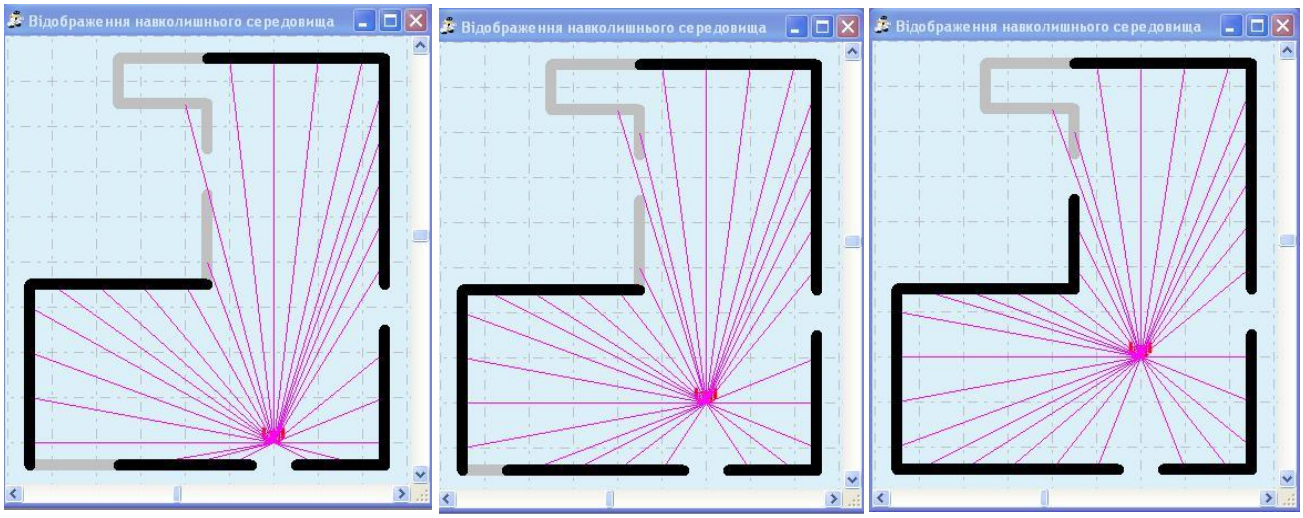


Рис. 8. Процес добудови семантичної мапи агентом у процесі його руху

Процес створення агентом семантичної мапи аналізованого приміщення завершується тоді, коли агент вже не може додати нової інформації до мапи. Зауважимо, що промені на рис. 3, 5 та 8 показано умовно, лише для ілюстрації особливостей процесу розпізнавання навколишнього середовища.

5. Методи розпізнавання кутів приміщень в навколишньому середовищі

Як зазначено в [1], кути приміщень можна умовно розподілити на два види: внутрішні (відповідають так званим «глухим» кутам у приміщенні, головною ознакою яких є те, що вони унеможливлюють подальший рух агента в їх напрямку) та зовнішні (головною ознакою яких є те, що кожний з них є кутом деякого отвору, через який агент може перейти в інше приміщення або в іншу частину даного приміщення). Відповідно до цього розроблено методи розпізнавання цих двох видів кутів.

Для зручності викладення цих методів розглянемо приклад приміщення на рис. 9, в якому агент розташований у деякій точці A і «спостерігає» навколишнє середовище, направляючи в нього промені з певним змінним кроком.

Методи ґрунтуються на наступних правилах прийняття рішень агентом:

1. Агент розпізнає кут приміщення як ***внутрішній***, якщо виконується наступна перевірка (рис. 9): $AE > AE_1$, $AE > AE_2$ (ці співвідношення також справедливі для всіх інших внутрішніх кутів O, L, H).

2. Агент розпізнає кути приміщення як ***зовнішні***, якщо виконується одна з наступних перевірок (рис. 9):

2.1. *За умов повної видимості кута* (наприклад, кута B), *коли два відрізки при їх зведенні дають загальну координату* (наприклад, BB_1 та B_2B дають координату B) та виконується наступна перевірка: $AB < AB_1$, $AB < AB_2$ (ці співвідношення також справедливі для кутів D, F, G, J, M, N, P).

2.2. *За умов часткової видимості кута, коли два відрізки при їх зведенні не дають загальної координати*, можливі два варіанти:

2.2.1. Коли точки перетинів належать прямим, що *перпендикулярні* одна одній (наприклад, точки C_1 та C_2 при визначенні отвору C – рис. 9), то виконується перевірка: $AC > AC_1$, $AC < AC_2$ (ці співвідношення також справедливі для кута K та кутів S, T на рис. 10). Зауважимо, якщо є тільки дві точки (C, C_1) і третій промінь не дав точки перетину, то AC_2 приймається як нескінченність, і ця перевірка також виконується.

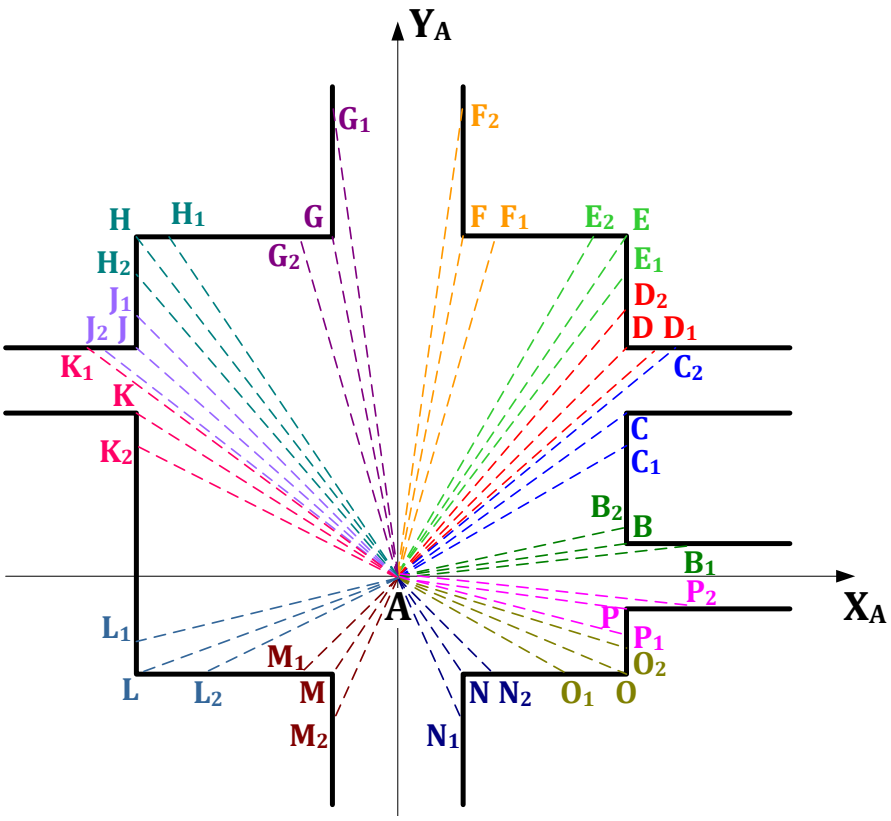


Рис. 9. Схема розпізнавання кутів приміщення агентом, розташованим у точці А

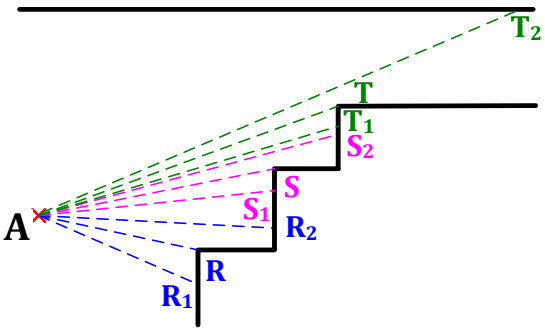


Рис. 10. Фрагмент схеми розпізнавання кутів агентом, розташованим у точці А

2.2.2. Коли точки перетинів належать прямим, що *паралельні* одна одній (наприклад, точки R_1 та R_2 при визначенні отвору R – рис. 10), то виконується перевірка $AR < AR_1$, $AR < AR_2$.

Висновки

Реалізація та використання запропонованих методів в МАС САГА показало досить високу їх ефективність для вирішення задачі розпізнавання агентом невідомого навколишнього середовища. Незважаючи на це, процедура розпізнавання має певні недоліки, усунення яких потребує проведення додаткових досліджень.

Зокрема, виходячи з цієї процедури, агент спроможний розпізнати стіну, якщо має можливість проаналізувати хоча б дві послідовно розташовані точки, що їй належать. Водночас в реальних обставинах у нього такої можливості може не бути. Наприклад, на рис. 11 показано випадки (фрагменти мапи, обведені колами), коли агент, використовуючи запропоновану процедуру розпізнавання, ніколи не зможе завершити формування

семантичної мапи такого приміщення, оскільки (де б він не знаходився) в нього завжди буде не вистачати необхідної інформації для завершення процесу розпізнавання.

Крім того, процес розпізнавання отворів у приміщеннях, що засновується на методах розпізнавання кутів приміщення, теж може давати похибки. Наприклад (рис. 12), агент, користуючись правилами прийняття рішень, викладеними в п. 5 статті, не зможе коректно визначити кут Q, оскільки ця ситуація (коли точки перетинів належать прямим, що *перпендикулярні* одна одній) відповідає п. 2.2.1 правил, за яким має бути $AQ > AQ_1$, $AQ < AQ_2$, водночас як за фактом: $AQ < AQ_1$, $AQ < AQ_2$.

Отже, запропоновані методи потребують відповідних удосконалень.

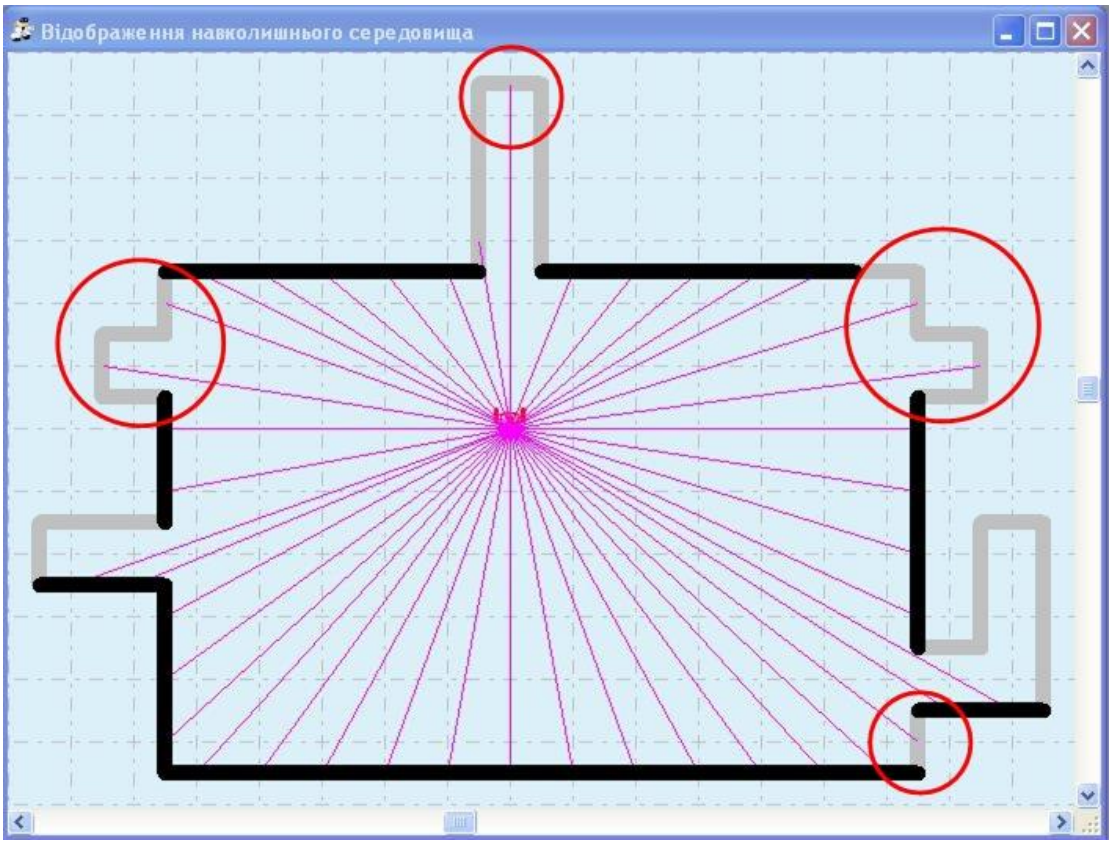


Рис. 11. Приклади відсутності інформації, необхідної для успішного розпізнавання

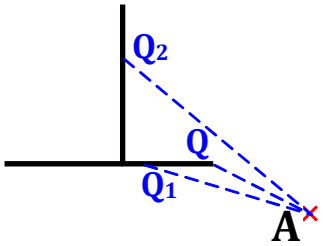


Рис. 12. Фрагмент схеми розпізнавання кута агентом, розташованим у точці A

Література

1. Яловець А.Л. До постановки задачі розпізнавання невідомого оточуючого середовища, навігації та планування шляхів агентом в ньому. *Проблеми програмування.* 2018. № 1. С. 113–127.
2. Nüchter A., Hertzberg J. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*. 2008. Vol. 56, N 11. P. 915–926.

References

1. Yalovets A.L. Statement of problem of unknown environment recognizing, navigating and path planning by agent. *Problems in programming.* 2018. N 1. P. 113–127 (in Ukrainian).
2. Nüchter A., Hertzberg J. Towards semantic maps for mobile robots. Robotics and Autonomous Systems. 2008. Vol. 56, N 11. P. 915–926.

Одержано 11.02.2019

Про автора:

Яловець Андрій Леонідович,
доктор технічних наук,
заступник директора інституту.
Кількість наукових публікацій в українських виданнях – понад 100.
Кількість наукових публікацій в зарубіжних виданнях – 10.
http://orcid.org/0000-0001-6542-3483.

Місце роботи автора:

Інститут програмних систем
НАН України.
03187, Київ-187,
проспект Академіка Глушкова, 40.

Тел.: (044) 526 15 38.
E-mail: yal@isofts.kiev.ua

УДК 612.51.001.57+519.6 https://doi.org/10.15407/pp2019.01.090

Р.Д. Григорян, А.Г. Дегода, В.С. Харсун, Е.А. Джуринский

СИМУЛЯТОР МЕХАНИЗМОВ СРОЧНОЙ РЕГУЛЯЦИИ ГЕМОДИНАМИКИ ЧЕЛОВЕКА

Создан программный симулятор (ПС) физиологических механизмов срочной регуляции гемодинамики человека. ПС основан на математической модели (ММ), описывающей рефлексы, источником информации, расположенных в правом желудочке сердца, в дуге аорты и в каротидных синусах механорецепторы. В качестве объекта управления (ОУ) используется ранее разработанная модель неуправляемой сердечно-сосудистой системы с пульсирующим сердцем. В ММ мишень регуляторных воздействий – длительность сердечного цикла, жесткость и ненапряженный объем сосудистых участков тела. Комплекс ММ и ОУ реализован на C^{++}. Ориентированный на физиолога интерфейс предоставляет ему возможность включения/выключения любого из рефлексов, проведения тестовых исследований (в том числе имитацию дозированной кровопотери или переливания крови). ПС пока функционирует автономно. В дальнейшем, после моделирования также эндокринных физиологических механизмов долговременного влияния на гемодинамику, ПС станет виртуальным средством для исследований комплексных механизмов оптимизации гемодинамики человека.

Ключевые слова: математическая модель, физиология, артериальное давление, рефлекторная регуляция, информационная технология.

Введение

Срочная регуляция гемодинамики охватывает временной интервал от секунд до нескольких минут. Существуют несколько специализированных регуляторов, эффекторами которых является исключительно сердечно-сосудистая система (ССС) [1]. Есть также рефлексы, в которых наряду с ССС эффекторами выступают легкие (например, хеморецепторные) или почки [1, 2]. Главными быстрыми регуляторами состояния ССС являются артериальные барорецепторные рфлексы (АБР). Наибольшие скопления рецепторов локализованы в дуге аорты и в каротидных синусах. По сути, барорецепторы – это механорецепторы, реагирующие на растяжение сосуда. Механорецепторами богат и миокард: в каждой камере сердца имеется свое скопление рецепторов. Они отличаются порогами срабатывания и уровнями насыщения.

Наиболее часто моделировались гемодинамические эффекты АБР [3–9]. Однако в этих моделях недостаточно адекватно описывались либо насосная функция сердца, либо взаимодействие рефлексов.

В реальном организме имеет место комплексная регуляция гемодинамики и упомянутыми рефлексами, и рядом нерефлекторных механизмов эндокринной природы [1, 2]. Но до настоящего времени отсутствует модель, позволяющая исследовать взаимодействие этих разных по своей природе и динамическим характеристикам механизмов регуляции гемодинамики в широких временных интервалах.

Наша цель – создать модель с возможно максимальным охватом известных физиологических регуляторов ССС. Для этого необходимо сделать несколько шагов: 1) создать модель объекта регулирования (эта модель уже разработана [8]); 2) создать модель механизмов срочной регуляции ССС; 3) создать модель нерефлекторных механизмов регуляции ССС; 4) объединить все модели в единый программно-моделирующий комплекс.

Цель данной работы описать модель, обозначенную в п. 2.

Основные требования к модели: допущения и ограничения

Моделируется аддитивное действие трех механорецепторных рефлексов, афферентные зоны, находящихся в дуге аорты, в каротидных синусах и в правом желудочке сердца.

Согласно физиологическим данным [10–12], клетки синусового узла, задающие ритм работы сердца, имеют спонтанную

активность. Ее модулируют температура и химический состав крови. Этот состав формируется как многими продуктами метаболизма клеток тела, также специальными медиаторами, выделяющихся из пресинаптических бляшек симпатичных и парасимпатических нервных волокон сердца. Основной медиатор симпатических нервов – это адреналин, а основным медиатором парасимпатических нервов – ацетилхолин. Скорость накопления каждого медиатора в синаптической щели пропорциональна частоте эфферентных импульсов.

1.1. *Формализация действия физиологических регуляторов.* Вначале формализуем функцию рецепторов. Опишем зависимость частоты афферентных импульсов в многоволоконном нерве (аортального, синусового или кардиального) ($N_j(t)$) от динамики входного фактора, действующего на рецептор. Для артериальных механорецепторов действующим на рецептор фактором является переменное во времени давление ($P_j(t)$). В естественных условиях это давление порождено давлением крови в артериях или давлением, созданным сердечной мышцей. Рецептор имеет порог возбуждения (P_j^*) и уровень насыщения (P_j^{**}). Между этими экстремальными значениями входного фактора зависимость между $N_j(t)$ и $P_j(t) - P_j^*$ имеет нелинейный вид, по форме напоминающий английскую букву «S». Анализ этой зависимости в физиологических условиях и подходов к ее моделированию убедил нас в том, что в общем случае функцию рецепторов следует моделировать с помощью статической зависимости:

$$N_j(t) = \begin{cases} 0, & 0 \le P_j(t) \le P_j^* \\ \dfrac{1 - e^{\alpha_j (P_j^* - P_j(t))}}{1 + \beta_j e^{\alpha_j (P_j^* - P_j(t))}}, & P_j^* < P_j(t) < P_j^{**} \\ 1, & P_j(t) \ge P_j^{**} \end{cases}. \quad (1)$$

Константы α_j и β_j – дополнительные характеристики аппроксимации, избирающие на основе эмпирики, $j = \overline{1,3}$. Изменения числовых значений этих констант позволяют имитировать изменения чувствительности рецепторов к колебаниям давления крови или механического усилия, развиваемого миокардом.

Согласно аддитивности рефлексов,

$$I_{\sum A}(t) = \sum_{j=1}^{3} a_j N_j(t), \text{ где } \sum_{j=1}^{3} a_j = 1. \quad (2)$$

Предположение 1. Химический состав крови ($X_S(t)$), поддерживающий деятельность нейронов синусового узла сердца неизменным ($X_S(t) = X_{SR}$).

Предположение 2. Температура крови ($T^o(t)$) также постоянная ($T^o(t) = T^o{}_0$).

Предположение 3. Самостоятельным модулятором активности эфферентных нейронов симпатического ($E_{ES}(t)$) и парасимпатического центров ($E_{EV}(t)$) головного мозга является суммарная его активность ($I_{BE}(t)$).

Предположение 4. Вместо описания трансформации активности эфферентных нейронов симпатического ($E_{ES}(t)$) и парасимпатического центров ($E_{EV}(t)$) в соответствующие медиаторы, непосредственно свяжем изменения параметров сердца и сосудов с $I_{BE}(t)$ и $I_{\sum A}(t)$.

$$E_{ES}(t) = \begin{cases} E_{ES}^{\min}, & E_{ES}(t) < E_{ES}^{\min} \\ \delta \cdot I_{BE}(t) + \varepsilon / I_{\sum A}(t), & E_{ES}^{\min} \le E_{ES}(t) \le E_{ES}^{\max} \\ E_{ES}^{\max}, & E_{ES}(t) > E_{ES}^{\max} \end{cases}, \quad (3)$$

$$E_{EV}(t) = \begin{cases} E_{EV}^{\min}, & E_{EV}(t) < E_{EV}^{\min} \\ \lambda \cdot I_{\sum A}(t) - \eta \cdot I_{BE}(t), & E_{EV}^{\min} \le E_{EV}(t) \le E_{EV}^{\max} \\ E_{EV}^{\max}, & E_{ES}(t) > E_{EV}^{\max} \end{cases}, \quad (4)$$

где $\delta, \varepsilon, \lambda, \eta$ – константы аппроксимации.

В данной статье принимается $I_{BE}(t) = 0$.

$E_{ES}(t)$ и $E_{EV}(t)$ представляют общую эфферентную активность многоволоконных нервов. Эти нервы имеют сердечные и сосудистые ветви.

Особенности динамики нервной регуляции длины сердечного цикла моделируем с помощью следующих двух дифференциальных уравнений:

$$E_{HS}(t) =$$

$$= \begin{cases} E_{HS}^{\min}, & E_{HS}(t) < E_{HS}^{\min} \\ \delta_H \cdot I_{BE}(t) + \varepsilon_H / I_{\sum A}(t), & E_{HS}^{\min} \le E_{HS}(t) \le E_{HS}^{\max} \\ E_{HS}^{\max}, & E_{HS}(t) > E_{HS}^{\max} \end{cases}, \quad (5)$$

$$E_{HV}(t) =$$

$$= \begin{cases} E_{HV}^{\min}, & E_{HV}(t) < E_{HV}^{\min} \\ \lambda_H \cdot I_{BE}(t) - \eta_H \cdot I_{BE}(t), & E_{HV}^{\min} \le E_{HV}(t) \le E_{HV}^{\max} \\ E_{HV}^{\max}, & E_{HS}(t) > E_{HV}^{\max} \end{cases}, \quad (6)$$

В уравнениях (5) и (6) константы δ_H, ε_H и λ_H, η_H позволяют моделировать чувствительность $E_{ES}(t)$ и $E_{EV}(t)$ к изменениям всех трех их детерминантов.

Об особенностях изменения $T_{Hc}(t)$.

$T_{Hc}(t)$ регулируется сердечными ветвями симпатического и парасимпатического нервов. Поскольку сердце функционирует как дискретный элемент, а рефлексы с зон правого желудочка и артериальных механорецепторов активируются в различных фазах сердечного цикла, необходима четкая дифференциация этих рефлексов в модели. Афферентная информация о результатах растяжения правого желудочка сердца формируется в текущем (k) цикле работы сердца и поступает в продолговатый мозг в текущей диастоле. Афферентная информация о результатах растяжения дуги аорты и каротидных синусов формируется только в фазе систолы, а именно, после начала фазы изгнания. Время задержки импульсов в пути к нейронам ядра продолговатого центра, на их центральную обработку, а также на процессы формирования регулирующих нервных изменений и на высвобождение биологически активных медиаторов такие, что эти изменения могут повлиять только на продолжительности $(k+1)$-го сердечного цикла.

Обозначим средние величины эфферентных симпатических и парасимпатических импульсов k-го сердечного цикла $\Delta E_{HS}(k)$ и $\Delta E_{HV}(k)$, соответственно. В данной модели допустим описание промежуточного звена, превращает частоту эфферентных симпатических и парасимпатических импульсов в химические медиаторы адреналина и ацетилхолина. Тогда $\Delta E_{HS}(k)$ и $\Delta E_{HV}(k)$ можно вычислять так:

$$\Delta E_{HS}(k) = \frac{1}{T_{Hc}(k)} \int_0^{T_{Hc}(k)} E_{HS}(t)dt,$$
$$\Delta E_{HV}(k) = \frac{1}{T_{Hc}(k)} \int_0^{T_{Hc}(k)} E_{HV}(t))dt, \quad (7)$$

$$E_{HS}(k+1) = E_{HS}0 + \Delta E_{HS}(k), \quad (8)$$

$$E_{HV}(k+1) = E_{HV}0 + \Delta E_{HV}(k), \quad (9)$$

где T_{Ac} – длина сердечного цикла в режиме автоматизма, μ, ϑ и χ – коэффициенты чувствительности $T_{Hc}(k+1)$ к изменениям соответствующих факторов, с помощью $\Delta E_{HS}(k)$ и $\Delta E_{HV}(k)$ модифицируют $T_{Hc}(k+1)$.

$$T_{Hc}(k+1) = T_{Ac} + \mu \cdot E_{HV}(k+1) -$$
$$- \vartheta \cdot E_{HS}(k+1) - \chi \cdot (T^o(t) - T^o{}_0). \quad (10)$$

Длины фаз диастолы ($T_d(k+1)$) и систолы ($T_s(k+1)$) вычисляем как:

$$T_s(k+1) =$$

$$= \begin{cases} 0.1, & T_c(k+1) < 0.25 \\ 0.4 \cdot T_c(k+1) + 0.1, & 0.25 \le T_c(k+1) \le 1.25 \\ 0.5, & T_c(k+1) > 1.25 \end{cases}; \quad (11)$$

$$T_d(k+1) = T_c(k+1) - T_s(k+1). \qquad (12)$$

Итак, система уравнений $(1)-(12)$ представляет математическую модель срочной регуляции ССС человека. $I_{\sum A}(t)$ принимает значения по шкале (0,1). Диапазон изменений для $I_{BE}(t)$ принят (0 ± 1). То есть, сигналы, возбуждающие нейроны продолговатого мозга, меняются в диапазоне от нуля до 1, в то время, как тормозные сигналы изменяются в диапазоне от нуля до (–1). Поскольку детальной модели $I_{BE}(t)$ еще не имеем, влияние этого фактора на срочный рефлекс будем исследовать путем предоставления дискретных значений дискретно $I_{BE}(t) = -0.75;\ -0.5;\ -0.25;\ 0;\ 0.25;\ 0.5;\ 0.75$.

Влияние нервной регуляции на сосуды зачислены путем деления общего симпатического тонуса на региональные звена сосудов в соответствии с их представлением в модели неуправляемой гемодинамики (МНГ). Обозначаем симпатичной изменения жесткости $\Delta D_i(t)$, а ненапряженного объема сосудов – соответственно $\Delta U_i(t)$. Сначала считаем изменения общей активности сипматического сосудистого нерва как:

$$E_{SV}(t) =$$

$$= \begin{cases} E_{SV}^{\min}, & E_{SV}(t) < E_{SV}^{\min} \\ \delta \cdot I_{BE}(t) + \varepsilon / I_{\sum A}(t), E_{SV} \le E_{SV}(t) \le E_{SV}^{\max} \\ E_{SV}^{\max}, & E_{SV}(t) > E_{SV} \end{cases}, \qquad (13)$$

$$\Delta E_{SV}(t) = E_{SV}(t) - E_{SV0}. \qquad (14)$$

Далее, распределяем $\Delta E_{SV}(t)$ на каждый участок сосудов с помощью коэффициентов $\varphi_i \cdot \Delta E_{SV}(t)$. Но расстояние до различных участков разный. Чтобы учесть задержку времени на пути следования управляющих импульсов, связь между $\Delta E_{SVi}(t)$ и $\Delta E_{SV}(t)$ организовано через дифференциальное уравнение:

$$\tau_i \frac{d\Delta E_{SVi}(t)}{dt} = \varphi_i \cdot \Delta E_{SV}(t). \qquad (15)$$

Увеличение жесткости сосудов и уменьшение ненапряженного объема от их начальных значений D_{i0} и U_{i0} рассчитываются как:

$$D_i(t) = D_{i0} + q_i \cdot \Delta E_{SVi}(t), \qquad (16)$$

$$U_i(t) = U_{i0} - \gamma_l \cdot \Delta E_{SVi}(t), \qquad (17)$$

где φ_i, q и γ_l характеризует плотность соответствующих нервных окончаний в данном регионе сосудов.

Итак, система уравнений $(1)-(17)$ является полной базовой моделью срочной регуляции гемодинамики человека. Эта модель описывает реакции регуляторов на колебания сердечной и сосудистой гемодинамики как в течение каждого сердечного цикла, а также при применении дополнительных воздействий на гемодинамику.

Для симуляций гемодинамики на компьютере была создана специальная программа на языке С.

Сведения о программе

Технические особенности программы.

1. Исходный код написан на языке С99.

2. Используются только такие внешние компоненты, на которых нет лицензионных ограничений.

3. Использование только необходимого минимума внешних зависимостей:

• Nuklear – библиотека, состоящая из одного заголовочного файла. Основная ее задача – отображение графических фигур и компонентов интерфейса, а также предоставление интерактивных точек взаимодействия с ними. В данной программе для построения интерфейса используется модифицированная версия, что упрощает использование различных драйверов;

• math.h – библиотека, реализующая стандартные математические функции;

• Plplot – библиотека для построения графиков.

4. Использование процедурной парадигмы программирования.

5. Оптимизированное построение алгоритмов.

6. Использование архитектурной модели MVC (Model View Controller), с четким разграничением компонентов.

7. Для компиляции используется компилятор Clang, на основе LLVM, что позволяет повысить качество программы.

8. Кроссплатформенность на основных операционных системах (Linux, Mac, Windows), а также возможность имплементации на других платформах, с минимальными изменениями.

Описание физического представления данных. Для хранения данных используются файлы с собственным бинарным представлением. Для экономии ресурсов, обеспечения кросплатформенности и уменьшения времени расчетов используются следующие архитектурные решения:

- названия элементов не записываются в сам файл, а используются как имя файла;
- только первый бинарный блок элемента содержит в себе все данные. Следующие блоки содержат только данные, меняются;
- для хранения метеопараметров используется конфигурационный текстовый файл;
- необходимо, чтобы все файлы находились в одной директории.

Для управления данными существует компонент I/O, обеспечивающий запись/считывание всех представленных элементов.

Физическое представление состоит из таких элементов как:

- vein – общее представление параметров для базовых вен и артерий, содержащее в себе параметры давления, жесткости и другие параметры, являющиеся общими для всех производных элементов. Хранятся в файлах с меткой v_;
- ventricle – представление параметров желудочков сердца. Производное от vein (путем агрегации). Хранятся в файлах с меткой h_;
- regulated_vein – представление дополнительных параметров для регулируемой модели. Производное от vein (путем агрегации). Хранятся в файлах с меткой rv_;
- flow – представление параметров связей (потоков) между другими элементами. Хранятся в файлах с меткой f_;
- метаданные – представление параметров времени, карты связей и других необходимых данных. Сохраняются в текстовом файле конфигурции.

Описание вычислительного модуля имеет одну точку входа (процедуру) в файле System.h. В качестве аргумента в эту функцию передается путь в конфигурационный файл. Затем в памяти создаются экземпляры для каждого из элементов.

Компьютерный эксперимент начинается с установки его сценария и задания текущих параметров модели (ее версию, характер внешнего воздействия, время экспозиции). Для решения уравнений модели используется метод трапеций.

По мере вычисления, состояние каждого элемента записывается в соответствующий результирующий файл для последующего отображения. Для экономии ресурсов – записывается не каждое исчисленное состояние элементов, а только в каждый шаг записи, заданный пользователем.

Основным способом анализа полученных результатов компьютерного эксперимента является визуальный анализ динамики интересующихся гемодинамических характеристик. Для визуализации выбранных посредством пользовательского интерфейса характеристик строятся их графики.

Вид пользовательского интерфейса, ориентированного на физиолога-исследователя, показан на рис. 1.

Интерфейс позволяет включить/выключить контур регуляции, установить длительность симуляционного эксперимента и запустить программу на исполнение. По завершении эксперимента пользователь на интерфейсе выбирает одну или более характеристик гемодинамики и активирует кнопку их визуализации в виде графиков.

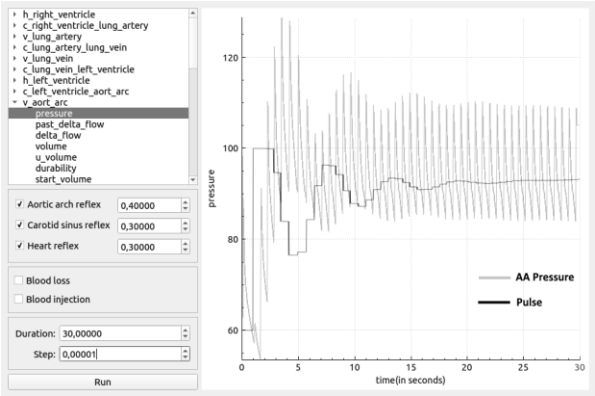


Рис. 1. Интерфейс пользователя

Тестовые результаты симуляций и их обсуждение

Прежде чем рассмотреть результаты тестовых симуляций отметим, что общий объем крови ($V_S = 5300\,cm^3$) и численные значения параметров модели ССС соответствуют здоровому молодому мужчине массой тела 70 кг. Полагалось, что в норме частота сокращений сердца равна $F = 60\ \mathrm{min}^{-1}$, среднее артериальное давление равно 94 мм рт. ст., (это соответствует пиковым значениям пульсового давления 120/80), общее периферическое сопротивление составляет 1 мм рт. ст. х сек/см^3.

На рис. 2. показаны динамика давления в дуге аорты (P_A), частоты сокращений сердца F в контрольных условиях, имитирующих покой в горизонтальном положении тела человека без каких-либо внешних воздействий. Как видно из отображенных на этом рисунке кривых, циклические изменения P_A характерны для условий физиологической нормы. Параметры регуляторов подобраны таким образом, чтобы каждый из моделируемых трех рефлексов вносил 1/3 суммарного рефлекторного ответа механизма нервной регуляции на изменения афферентной импульсации, приходящей из рецепторов правого желудочка, дуги аорты и каротидных синусов. Теоретически, эти весовые соотношения могут быть изменены в каждом эксперименте, что дает возможность имитировать влияние усиления (ослабления) вклада любого из них в формирование суммарного рефлекторного ответа на кардиоциклические возмущения.

Для иллюстрации рефлекторного контроля гемодинамики на более длительных отрезках времени было сымитировано двухстороннее изменение общего объема крови. Эти результаты показаны на рис. 3 и 4. На этих рисунках показан результат симуляции кровопотери со скоростью 20 мл/сек до достижения максимальной величины кровопотери 400 мл.

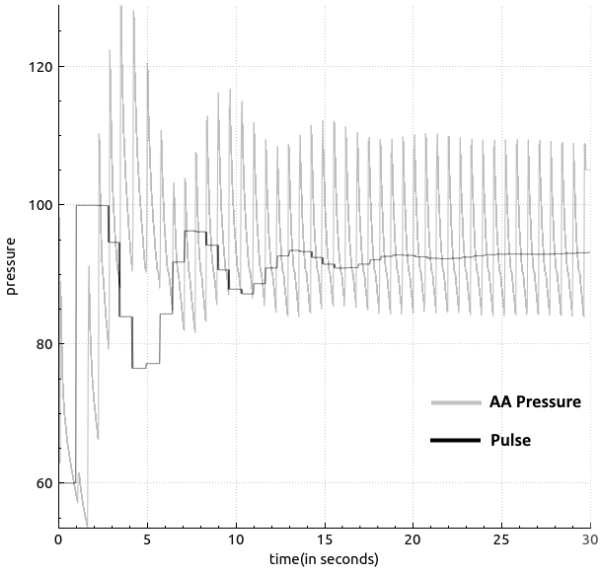


Рис. 2. Динамика давления в дуге аорты (P_A) и частоты сокращений сердца F в модели рефлекторно управляемой ССС в контрольных условиях

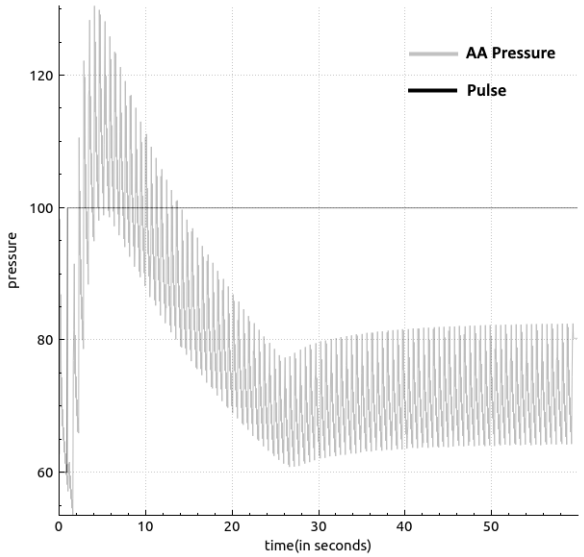


Рис. 3. Динамика давления в дуге аорты (P_A) и частоты сокращений сердца F в модели неуправляемой ССС 500 мл (со скоростью 20 мл/сек)

Разница в том, что в случае, показанном на рис. 3, эта процедура имитировалась на модели неуправляемой ССС, а на рис. 4 показан результат моделирования той же процедуры на модели с включенными рефлексами. Сравнение картинок на этих рисунках рельефно иллюстрирует системную роль регуляторов: она состоит в компенсации падения P_A, вызванного уменьшением центрального венозного давления и, соответственно, притока крови к сердцу.

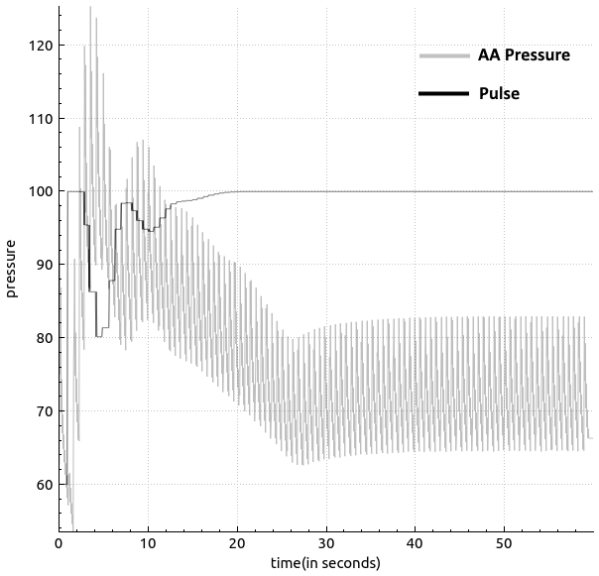


Рис. 4. Динамика давления в дуге аорты (P_A) и частоты сокращений сердца *F* в модели рефлекторно управляемой ССС при имитации дозированной кровопотери 500 мл (со скоростью 20 мл/сек)

Помимо результатов, представленных на рис. 2, 3 и 4, проведены также другие тестовые симуляции. В целом, результаты тестовых экспериментов показали, что уравнения модели срочной регуляции адекватно описывают функцию естественных физиологических механизмов, которые срочно реагируют на внутренние/внешние возмущения состояния ССС. Что касается констант, моделирующих пороги рецепции, чувствительности каждого из рефлексов, а также предельные значения их функционирования, они также подобраны достаточно адекватно.

Выводы

Создана и тестирована базовая математическая модель механизмов срочной рефлекторной регуляции гемодинамики человека. Реализованная в виде автономной программы на языке C^{++} модель является симулятором, позволяющим имитировать реакции гемодинамики на пульсации сердца, а также на экзогенные воздействия. Модель станет компонентом в разрабатываемой комплексной модели нейроэндокринных влияний на состояние сердечно-сосудистой системы человека.

Литература

1. Brands M.W. Chronic Blood Pressure Control. Compr. Physiol. 2012. Vol. 2. P. 2481–2494.
2. Grygoryan R.D. The optimal circulation: cells' contribution to arterial pressure. 2017. Nova Science. N.Y. 298 p.
3. Григорян Р.Д., Лиссов П.Н. Программный имитатор сердечно-сосудистой системы человека на основе ее математической модели. *Проблеми програмування*. 2004. № 4. С. 100–111.
4. Григорян Р.Д., Лиссов П.Н., Аксенова Т.В., Мороз А.Г. Специализированный программно-моделирующий комплекс «PhysiolResp». *Проблеми програмування.* 2009. № 2. С. 140–150.
5. Larrabide I., Blanco P.J., Urquiza S.A., Dari E.A., Ve´nere M.J., de Souza e Silva N.A., Feijo´ R.A. HeMoLab – Hemodynamics Modelling Laboratory: An application for modelling the human cardiovascular system. Computers in Biology and Medicine. 2012. Vol. 42. P. 993–1004.
6. Ježe F., Kulhánek T., Kalecký K., Kofránek J. Lumped models of the cardiovascular system of various complexity. *Biocybernetics and Biomedical Engineering*. 2017. Vol. 37. Issue 4. P. 666–678.
7. Kulhánek T., Kofránek J., Mateják M. Modeling of short-term mechanism of arterial pressure control in the cardiovascular system: Object-oriented and acausal approach. *Computers in Biology and Medicine*. 2014. Vol. 54. P. 137–144.
8. Григорян Р.Д., Дегода А.Г., Джуринский Е.А., Харсун В.С. Симулятор пульсирую-

щего сердца. *Проблеми програмування*. 2017. № 4. С. 98–108.
9. Mahdi A., Sturdy J., Ottesen J.T., Olufsen M.S. Modeling the Afferent Dynamics of the Baroreflex Control System. *PLoS Comput Biol*. 2013 Dec; 9(12): e1003384.
10. Raphan T., Cohen B., Xiang Y., Yakushin S. B. A Model of Blood Pressure, Heart Rate, and Vaso-Vagal Responses Produced by Vestibulo-Sympathetic Activation. Front Neurosci. 2016. Vol. 10. P. 96–104.
11. Stewart J.M. Update on the theory and management of orthostatic intolerance and related syndromes in adolescents and children. *Expert Rev. Cardiovasc. Ther.* 2012. Vol. 10(11), P. 1387–1399.
12. Kawada T., Yamamoto K., Kamiya A., Ariumi H., Michikami D., Shishido T., Sunagawa K., Sugimachi M. A derivative-sigmoidal model reproduces operating point-dependent baroreflex neural arc transfer characteristics. *Jpn. J. Physiol*. 2005. Vol. 55(3), P. 157–163.

References

1. Brands M. W. Chronic Blood Pressure Control. Compr. Physiol., 2012. Vol. 2. P. 2481–2494.
2. Grygoryan R.D. The optimal circulation: cells' contribution to arterial pressure. 2017, Nova Science, N.Y., 298 p.
3. Grygoryan R.D., Lissov P.N. A software imitator of human cardiovascular system basen on its mathematical model. Problems in programming. 2004, N 4. P. 100–111.
4. Grygoryan R.D., Lissov P.N., Aksionova T.V., Moroz A.G. A specialized software-modeling complex «PhysiolResp». Problems in programming. 2009, N 2. P. 140–150.
5. Larrabide I., Blanco P.J., Urquiza S.A., Dari E.A., Ve´nere M.J., de Souza e Silva N.A., Feijo´ R.A. HeMoLab – Hemodynamics Modelling Laboratory: An application for modelling the human cardiovascular system. Computers in Biology and Medicine. 2012. Vol. 42. P. 993–1004.
6. Ježe F., Kulhánek T.,Kalecký K., Kofránek J. Lumped models of the cardiovascular system of various complexity. Biocybernetics and Biomedical Engineering. 2017. Vol. 37. Issue 4, P. 666–678.
7. Kulhánek T., Kofránek J. , Mateják M. Modeling of short-term mechanism of arterial pressure control in the cardiovascular system: Object-oriented and acausal approach. Computers in Biology and Medicine, 2014. Vol. 54. P. 137–144.
8. Grygoryan R.D., Degoda A.G., Dzhurinsky E.A., Kharsun V.S. A simulator of the pulsatile heart. Problems in programming. 2017. N 4. P. 98–108.
9. Mahdi A., Sturdy J., Ottesen J.T., Olufsen M.S. Modeling the Afferent Dynamics of the Baroreflex Control System. PLoS Comput Biol. 2013, 9(12): e1003384.
10. Raphan T., Cohen B., Xiang Y., Yakushin S. B. A Model of Blood Pressure, Heart Rate, and Vaso-Vagal Responses Produced by Vestibulo-Sympathetic Activation. Front Neurosci. 2016. Vol. 10. P. 96–104.
11. Stewart J.M. Update on the theory and management of orthostatic intolerance and related syndromes in adolescents and children. Expert Rev. Cardiovasc. Ther. 2012. Vol. 10(11). P. 1387–1399.
12. Kawada T., Yamamoto K., Kamiya A., Ariumi H., Michikami D., Shishido T., Sunagawa K., Sugimachi M. A derivative-sigmoidal model reproduces operating point-dependent baroreflex neural arc transfer characteristics. Jpn. J. Physiol. 2005. Vol. 55(3). P. 157–163.

Получено 11.02.2019

Об авторах:

Григорян Рафик Давидович,
заведующий отделом,
доктор биологических наук.
Количество научных публикаций в украинских изданиях – 132.
Количество научных публикаций в зарубежных изданиях – 40.
Индекс Хирша – 9.
http://orcid.org/0000-0001-8762-733X,

Дегода Анна Григорьевна,
старший научный сотрудник,
кандидат физико-математических наук.
Количество научных публикаций в украинских изданиях – 10.

Количество научных публикаций в
зарубежных изданиях – 1.
Индекс Хирша – 3.
http://orcid.org/0000-0001-6364-5568,

Харсун Вадим Сергеевич,
инженер-программист.
Количество научных публикаций в
украинских изданиях – 1.
http://orcid.org/0000-0001-5745-0932,

Джуринский Егор Антонович,
инженер-программист.
Количество научных публикаций
в украинских изданиях – 1.
http://orcid.org/0000-0002-1636-1447.

Место работы авторов:

Институт программных систем
НАН Украины,
03187, Киев,
проспект Академика Глушкова, 40.
Тел.: (044) 526 5169.
E-mail: rgrygoryan@gmail.com,
anna@silverlinecrm.com,
vakharsun@gmail.com,
y.a.dzhurynskyi@gmail.com

УДК 004.42:510.69

UDC 004.42:510.69

Пропозициональные логики частичных предикатов с композицией предикатного дополнения / Н.С. Никитченко, О.С. Шкильняк, С.С. Шкильняк, Т.А. Мамедов. – С. 3–13.

Propositional logics of partial predicates with composition of predicate complement / M.S. Nikitchenko, O.S. Shkilniak, S.S. Shkilniak, T.A. Mamedov. – P. 3–13.

В работе исследованы новые программно-ориентированные логические формализмы – логики частичных предикатов с предикатным дополнением, названные LC. Характерная особенность таких логик – наличие специальной немонотонной операции (композиции) предикатного дополнения. Подобные операции используются в различных вариантах логик Флойда – Хоара с частичными пред- и после-условиями. Свойства пропозициональных композиций LC аналогичны свойствам традиционных логических связок. Исследованы свойства новой композиции предикатного дополнения. Класс P-предикатов (однозначных) замкнут относительно композиции предикатного дополнения, а класс T-предикатов (тотальных) незамкнут. Поэтому можно рассматривать общий класс LC – логики R-предикатов (реляционного типа) с композицией предикатного дополнения, и их подкласс LPC – это логики P-предикатов с такой композицией. В центре внимания работы – изучение пропозициональных LC, или PLC. Описаны пропозициональные композиционные алгебры и языки PLC. Для LC однозначных предикатов предложено и исследовано отношение |=IR⊥ неопровержимого логического следствия при условиях неопределенности. Приведены условия наличия отношения |=IR⊥ и свойства декомпозиции формул. На этой основе для PLC P-предикатов построено исчисление секвенциального типа. Приведены базовые секвенциальные формы этого исчисления и условия замкнутости секвенций. Для построенного исчисления верны теоре-

The paper studies new software-oriented logical formalisms – the logics of partial predicates with predicate complement. Such logics are denoted LC. A characteristic feature of these logics is the presence of a special non-monotonic operation (composition) of the predicate complement. Such operations are used in various versions of the Floyd-Hoare logic with partial pre- and post-conditions. Properties of LC propositional compositions are similar to the properties of the traditional logical connectives. Properties of the new composition of the predicate complement are investigated. The class of *P*-predicates (partial single-valued) is closed under the composition of the predicate complement, but the class of *T*-predicates (total) is not closed. Therefore, it is possible to consider the general class LC – the logic of *R*-predicates (relational predicates) with the composition of the predicate complement, and its subclass LPC – the logic of *P*-predicates with such a composition. The focus of the work is the study of PLC – propositional LC. Propositional composition algebras and PLC languages are described. For LC of partial single-valued predicates, an irrefutability logical consequence relation $|=_{IR}^{\perp}$ is proposed and investigated under the conditions of undefinedness. The conditions for the validity of the $|=_{IR}^{\perp}$ and the properties of the decomposition of formulas are given. Based on the properties of the $|=_{IR}^{\perp}$, for PLC of *P*-predicates a calculus of sequential type is constructed. The basic sequential forms of this calculus and closure conditions of the sequents are given. For the constructed calculus,

мы корректности и полноты. Доказательства этих теорем будут приведены в последующих статьях.

Ключевые слова: логика, частичный предикат, логическое следствие, секвенциальное исчисление.

УДК 004.42:510.69

Девиантные алгебры истинностных значений и девиантные классы общих недетерминованных предикатов / О.С. Шкильняк. – С. 14–26.

Изучаются новые классы программно-ориентированных логических формализмов – логики общих недетерминированных (GND) предикатов. GND-предикаты можно моделировать как 7-значные тотальные детерминированные (TD7) предикаты. Основное внимание уделено исследованию алгебр истинностных значений (TV-алгебр) TD7-предикатов. Существует 20 подалгебр TV-алгебры TD7-предикатов ATV7 = (TV7, $\{\neg^*, \vee^*\}$). Каждая из них индуцирует соответствующую алгебру TD7-предикатов. Существует очень много 7-значных логик, поэтому много подмножеств TV7 незамкнуты относительно $\neg^*$ или $\vee^*$, они не образуют подалгебр ATV7. Эти подмножества и соответствующие классы GND-предикатов названы девиантными. Для того, чтобы девиантное TV $\subseteq$ TV7 образовало алгебру, нужно модифицировать $\neg^*$ или $\vee^*$. Это можно сделать многими способами. Модификация $\neg^*$ ведет к специфическим неклассическим логикам и в этой работе не рассматривается. Особо важны такие модификации $\vee^*$, для которых выполняется условие TFC корректности логических связок предикатных алгебр. При нарушении TFC TV-алгебра названа девиантной, она не индуцирует алгебру GND-предикатов. Для всех подмножеств TV7 исследована возможность модификации $\vee^*$ с условием TFC, что определяет соответствующие классы GND-

correctness and completeness theorems are hold. Proofs of these theorems will be given in the forthcoming articles.

Key words: logic, partial predicate, logical consequence, sequent calculus.

UDC 004.42:510.69

Deviant truth-values algebras and deviant classes of general non-deterministic predicates / O.S. Shkilniak. – P. 14–26.

In this paper we study new classes of program-oriented logical formalisms – logics of general non-deterministic (GND) predicates. These logics reflect such properties of programs as nondeterminism, partiality, and non-fixed arity. GND-predicates can be modeled as 7-valued total deterministic (TD7) predicates. The main attention is paid to algebras of truth values (TV-algebras) of TD7-predicates. The set of truth values TV7 = {T, F, T↑, F↑, ↑, TF, TF↑} defines TV-algebra ATV7 = (TV7, $\{\neg^*, \vee^*\}$). There 20 subalgebras of ATV7 exist, and each of them induces a corresponding algebra of GND-predicates. At the same time there is a very large number of 7-valued logics so a lot of TV7 subsets are not closed under $\neg^*$ or $\vee^*$ and do not form subalgebras of ATV7. We call such subsets with the corresponding classes of GND-predicates deviant, they are not closed under logical connectives of GND-predicates. In order for deviant TV $\subseteq$ TV7 to form an algebra we need to modify $\neg^*$ or $\vee^*$. Modifications can be made in a large number of ways. Modification of $\neg^*$ leads to specific non-classical logics and lies outside the scope of this paper. Modifications of $\vee^*$ satisfying the TFC condition of predicate algebras logical connectives correctness are the most important, otherwise we obtain deviant TV-algebra which does not induce an algebra of GND-predicates. For all TV7 subsets we study the possibility of $\vee^*$ modifica-

предикатов. Описаны естественные модификации $\vee^*$ без TFC, что дает ряд девиантных TV-алгебр. Существуют 3 девиантных TV $\subseteq$ TV7, для которых нет модификаций $\vee^*$ с условием TFC; для них указаны относительно естественные девиантные TV-алгебры.

Ключевые слова: логика, алгебра, недетерминированный предикат, 7-значный предикат.

УДК 004.4'24

Автоматизированное проектирование OpenCL программ на основе алгебро-алгоритмического подхода / А.Е. Дорошенко, Н.Н. Бондаренко, Е.А. Яценко. – С. 27–36.

Дальнейший прогресс в улучшении показателей качества создания параллельных программ связан с использованием гетерогенных архитектур вычислительных систем. Неоднородные параллельные системы включают у себя, в частности, гибридные вычислительные платформы, сочетающие использование центральных процессоров и графических ускорителей. Одним из инструментальных средств программирования таких систем является OpenCL. В статье выполнена настройка ранее созданного алгебро-алгоритмического инструментария проектирования и синтеза на автоматизированную разработку OpenCL программ. Особенностью предложенного подхода к проектированию является использование высокоуровневого языка, основывающегося на системах алгоритмических алгебр Глушкова. Подход продемонстрирован на разработке программы интерполяции для задачи метеорологического прогнозирования. Проведен эксперимент по выполнению сгенерированной с помощью разработанного инструментария параллельной программы на графическом

tion with TFC condition. Such modification induces corresponding classes of GND-predicates. We describe “natural” modifications of $\vee^*$ without TFC condition obtaining a number of deviant TV-algebras. There are no modifications with TFC condition for deviant sets {↑, TF, TF↑}, {TF, ↑}, {TF↑, ↑}, so for them we specify “relatively natural” deviant TV-algebras.

Key words: logic, algebra, nondeterministic predicate, 7-valued predicate.

UDC 004.4'24

Automated design of OpenCL programs based on algebra-algorithmic approach / A.Yu. Doroshenko, M.M. Bondarenko, O.A. Yatsenko. – P. 27–36.

Further progress in improving the quality of parallel software development is linked to the use of heterogeneous architectures of parallel computing systems. Heterogeneous parallel systems, in particular, include hybrid computing platforms combining the use of central and graphics processing units. One of the facilities for programming such systems is OpenCL. The paper proposes the further development of previously developed algebra-algorithmic tools in the direction of automated design and synthesis of OpenCL programs. The particular feature of the proposed approach consists in using a high-level language based on Glushkov’s system of algorithmic algebra. The approach is illustrated on the development of a parallel interpolation algorithm, which is the part of the numerical weather forecasting program. The results of the experiment consisting in executing of the generated OpenCL program on a graphics processing unit are given. The program is compared with the implementation for CUDA platform.

Key words: automated software design,

ускорителе. Выполнено сравнение с реализацией программы на платформе CUDA.

Ключевые слова: автоматизированное проектирование программ, алгебра алгоритмов, метеорологическое прогнозирование, неоднородные параллельные вычислительные системы, синтез программ, CUDA, OpenCL.

УДК 004.4'2

К вопросу автоматизации проектирования рабочих процессов на основе алгебро-алгоритмического и онтологического инструментария / О.М. Овдей. – С. 37–47.

В статье представлена концепция системы для автоматизации проектирования рабочих процессов на основе алгебро-алгоритмического и онтологического инструментария. Разработанная ранее прикладная онтология для проектирования программ была расширена новыми концепциями, относящимися к проектированию рабочих процессов. Проведен анализ существующих подходов и предложена архитектура системы. Работу системы проиллюстрировано на примере разработки рабочего процесса Apache Oozie для анализа больших объемов данных на распределенной платформе Apache Hadoop. Показано, что сочетание инструментов онтологий и алгебро-алгоритмических инструментов обеспечивает значительный потенциал для адаптации, оптимизации, интеграции и модификации. Кроме того, обеспечиваемый данными инструментами высокий уровень абстракции облегчает понимание проектируемых рабочих процессов как людьми, так и программными агентами, упрощая совместную работу и способствуя распространению знаний. Результаты количественной оценки эффективности предложенных средств свидетельствуют о том, что они могут способствовать существенному повышению производ-

algorithm algebra, CUDA, heterogeneous parallel computing systems, meteorological forecasting, OpenCL, software synthesis.

UDC 004.4'2

On the issue of automating the workflow design based on algebra-algorithmic and ontological tools / O.M. Ovdii. – P. 37–47.

The article presents the concept of a system for automated workflow design based on algebra-algorithmic and ontological tools. The previously developed applied ontology for program design has been expanded with new concepts related to the workflow design. The existing approaches are analysed and architecture for system that under development is proposed. The work of the system is illustrated by the example of the development of the Apache Oozie workflow for analyzing large amounts of data on the distributed Apache Hadoop platform. It is shown that the combination of ontology and algebra-algorithmic tools provides significant potential for adaptation, optimization, integration and modification. In addition, the high level of abstraction provided by these tools facilitates the understanding of the designed workflows by both people and software agents, simplifying teamwork, workflow sharing and reuse. The results of a quantitative assessment of the proposed tools indicate that they can contribute to a significant increase in development productivity and reduce labor costs.

Key words: workflow, ontology, algebra of algorithms, program design and synthesis, distributed computing.

дительности разработки и уменьшению трудозатрат.

Ключевые слова: рабочие процессы, онтология, алгебра алгоритмов, проектирование и синтез программ, распределенные вычисления.

УДК 004.4'24

TermWare-3 – система переписывания термов, основанная на исчислении контекстов / Р.С. Шевченко, А.Е. Дорошенко. – С. 48–56.

В статье описывается конструкция системы переписывания термов TermWare-3, построенной на основе рефлексивного исчисления контекстных термов, когда структура терма включает в себя дополнительно к дереву терма еще внутренний контекст и ограничения на сопоставление (внешний контекст). Это позволяет погрузить операции разрешения имен в математическую семантику переписывающих правил. Элементами алгебры являются мультитермы, которые дополняют контекстные термы конструкциями стрелок, универсальным образцом сопоставления и непротиворечивых множеств. Это позволяет эмулировать подстановку типизированных переменных в рамках алгебры мультитермов. Также описывается метод эффективной диспетчеризации выбора правил. Приводится пример применения системы переписывающих правил для анализа смарт-контрактов вместе с автоматическим преобразованием выражения между системами на основе алгебраических типов Scala и контекстными термами.

Ключевые слова: автоматизированное проектирование программ, языки программирования, переписывание термов, системы типов, анализ кода, TermWare.

UDC 004.4'24

TermWare-3 – term rewriting system, based on context-term calculus /
R.S. Shevchenko, A.E. Doroshenko. – P. 48–56.

In this paper, the design of the TermWare-3 rewriting system which is built on the ground of reflective calculus of context term is considered. In the calculus of the context term, term structure contains not only tree, but a reference to an internal context and matching constrains (external context). This allows to embed operation of name resolving into the mathematical semantics of the term rewriting. The elements of algebra are multiterms, which complement context terms by constructions of arrows, universal matching pattern and consistent set. This allows to emulate substitutions of typed variables in the terms of multiterm algebra. Also, the method of effective rule dispatch is described. Example of usage term rewriting system for checking of smart-contract properties, with automatic transformation of expression between representation in terms of Scala algebraic types and TermWare-3 context terms
is shown.

Key words: automated software design, programming languages, term rewriting, type systems, code analysis, TermWare.

УДК 004.853, 004.55

Средства и методы анализа неструктурированных данных / Ю.В. Рогушина – С. 57–77.

Проанализированы современные средства анализа неструктурированных данных и влияние Big Data на актуальность этого направления исследований. Рассмотрены перспективы использования фоновых знаний для такого структурирования. Обоснована целесообразность применения для этого таких стандартов W3C, как RDF и OWL. Использование семантических Wiki-технологий для создания распределенных информационных ресурсов не только позволяет довольно легко добавлять структурирование к НСД, но и является источником фоновых знаний для анализа произвольных естественноязыковых текстов соответствующей предметной области. Предложенные в работе модели и методы позволяют усовершенствовать процесс генерации таких знаний.

Ключевые слова: неструктурированные данные, Text Mining, онтология, Semantic Web, Wiki.

UDC 004,853, 004.55

Means and methods of the unstructured data analysis / J. Rogushina. – P. 57–77.

Analysis of the current trends in the unstructured text data wide usage and the development of software tools for their processing causes the high urgency of this research direction and the necessity of intelligent information systems in such processing. A signigicant part of Big Data consists of unstructured texts that require the further development of specific Text Mining and algorythms of machine learning. Unstructured data consisting of natural language text in the general case, do not have a predetermined data model. Their ambiguity, heterogeneity and context dependence considerably complicate the classification of documents, the identification of their components and the automated obtaining of user-oriented knowledge from their content, while the large volumes and dynamism of such data do not involve efficient manual processing. The means and methods of data structuring, their various software implementations are considered. The prospects of using background knowledge for such structuring are analyzed. The feasibility of application such W3C standards as RDF and OWL is substantiated. The use of semantic Wiki-technologies for development of distributed information resources simplifies the process of natural text structuring by users and also generates the source of background knowledge for the analysis of arbitrary texts of the corresponding domains. The models and methods proposed in the work allow to improve this process.

Key words: unstructured data, ontology, Text Mining, Semantic Web, Wiki.

УДК 004.94+004.896

Методы распознавания агентом неизвестной окружающей среды /
А.Л. Яловец. – С. 78–89.

Изложены методы, используемые агентом для распознавания неизвестной окружающей среды, в том числе: метод определения расстояния до ближайшего видимого объекта и координат пересечения с этим объектом луча (как моделируемого направления зрения агента), направленного от агента; метод динамического изменения градации угла луча, направляемого агентом в окружающую среду; метод построения множества точек, принадлежащим видимым с точки текущего расположения агента объектам окружающей среды; метод обобщения агентом множества точек с построением фрагментов семантической карты неизвестной окружающей среды; методы распознавания углов помещений в окружающей среде.

Ключевые слова: агент, неизвестная окружающая среда, семантическая карта.

UDC 004.94+004.896

Methods of recognition by the agent of the unknown environment /
A.L. Yalovets. – P. 78–89.

The methods used by the agent to recognize an unknown environment are outlined. They include: a method for determining the distance to the nearest visible object and the coordinates of the intersection with this object of the beam (as a simulated direction of the agent's view) directed from the agent; the method of dynamically changing the gradation of the beam angle directed by the agent into the environment; a method for constructing a set of points belonging to objects of the environment visible from the point of the current location of the agent; the method of agent's summarizing a set of points with the construction of semantic map fragments of unknown environment; methods of recognition of the corners of rooms in the environment.

Key words: agent, unknown environment, semantic map.

УДК 612.51.001.57+519.6

Симулятор механізмів термінового регулювання гемодинаміки людини / Р.Д. Григорян, А.Г. Дегода, В.С. Харсун, Е.А. Джуринський. – С. 90–98.

Створено програмний симулятор (ПС) фізіологічних механізмів термінової регуляції гемодинаміки людини. ПС заснований на математичній моделі (ММ), яка описує рефлекси, джерелом інформації в яких є розташовані в правому шлуночку серця, в дузі аорти і в каротидних синусах механорецеп-

UDC 612.51.001.57+519.6

A simulator of mechanisms of acute control of human hemodynamics /
R.D. Grygoryan, A.G. Degoda, V.S. Kharsun, E.A. Dzhurinsky. – P. 90–98.

A software simulator (SS) of the physiological mechanisms that provide the acute control of human hemodynamics is created. SS is based on a mathematical model (MM) describing reflexes, the source of information in which are mechanoreceptors located in the right ventricle of the heart, in the aortic arch

тори. Як об'єкт управління (ОУ) використовується раніше розроблена модель некерованою серцево-судинної системи з пульсуючим серцем. В ММ мішенню регуляторних впливів є тривалість серцевого циклу, жорсткість і ненаголошений обсяг судинних ділянок тіла. Комплекс ММ і ОУ реалізований на C^{++}. Орієнтований на фізіолога інтерфейс надає йому можливість включення / вимикання будь-якого з рефлексів, проведення тестових досліджень (в тому числі імітацію дозованої крововтрати або переливання крові). ПС поки функціонує автономно. Надалі, після моделювання також ендокринних фізіологічних механізмів довгострокового впливу на гемодинаміку, ПС стане віртуальним засобом для досліджень комплексних механізмів оптимізації гемодинаміки людини.

Ключові слова: математична модель, фізіологія, артеріальний тиск, рефлекторна регуляція, інформаційна технологія.

and in the carotid sinuses. A previously developed model of an uncontrollable cardiovascular system with a pulsating heart is used as an object of control (OC). In MM, the duration of the cardiac cycle, the rigidity and the unstressed volume of the vascular areas of the body are the target of regulatory actions. MM and OC are realized in C++. Oriented to the physiologist, the interface provides him / her with the ability to turn on / off any of the reflexes, to conduct test studies (including imitation of dosed blood loss or blood transfusion). SS is still functions autonomously. Subsequently, after modeling also endocrine physiological mechanisms of long-term effects on hemodynamics, SS will become a virtual tool for research of complex mechanisms that optimize human hemodynamics.

Key words: mathematical model, physiology, blood pressure, reflex regulation, information technology.

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 до 16 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: tsok@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 – 11 пт. 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 на українській мові, то анотації та ключові слова – на російській та англійській мовах) 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**.