ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського
національного університету залізничного транспорту, 2021, № 4 (94)

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

УДК 004.622:[004.94:519.17]

О. С. КУРОП’ЯТНИК[1*], Б. М. ЯКОВЕНКО[2*]

[1*]Каф. «Комп’ютерні інформаційні технології», Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 373 15 35,
ел. пошта olena.kuropiatnyk@gmail.com, ORCID 0000-0003-2286-884X
[2*]Каф. «Комп’ютерні інформаційні технології», Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 373 15 35,
ел. пошта bohdanyakovenko98@gmail.com, ORCID 0000-0001-6174-0027

Визначення відповідності тексту алгоритму програми на основі конструктивно-продукційної моделі графа керування

Мета. Основною метою статті є розробка та програмна реалізація методу визначення відповідності тексту алгоритму програми, представленого у вигляді блок-схеми. **Методика.** Для зіставлення тексту програми та блок-схеми побудовано математичну модель їх перетворювачів у графове представлення з використанням апарату конструктивно-продукційного моделювання та його методів: спеціалізації, конкретизації, інтерпретації та реалізації. Графове представлення (графи) тексту будується з урахуванням операторів керування, блок-схеми – за json-файлом, що містить опис елементів схеми та їх зв’язків. Для порівняння графів застосовано метод пошуку в ширину з підрахунком кількості однакових вершин. Для програмної реалізації розробленого методу й моделей була застосована технологія об’єктно-орієнтованого програмування та CASE-технології, в основі яких лежить уніфікована мова моделювання UML. **Результати.** Запропоновано метод, що дозволяє представити текст та блок-схему програми в єдиному форматі орієнтованого графа (графа керування) та виконати оцінку їх відповідності за кількістю однакових вершин. Для його формалізації та автоматизованого використання розроблено конструктивно-продукційні моделі перетворювачів вхідних даних. На основі моделей та методу створено програмний додаток. **Наукова новизна.** Отримали подальший розвиток методи конструктивно-продукційного моделювання в задачах обробки текстів, написаних штучними мовами. Побудована система конструкторів, що виконує перетворення тексту програм мовою C++ у граф керування. **Практична значимість.** Результати роботи мають значення для розв’язання таких задач, як зіставлення текстів програм із метою виявлення запозичень, визначення відповідності алгоритмів програм їх програмним реалізаціям із метою поліпшення навичок кодування. Графове представлення, яке продукує розроблена система конструкторів, може бути застосоване для дослідження впливу оптимізації та рефакторингу коду на складність програм із використанням метрик МакКейба.

Ключові слова: конструктивно-продукційне моделювання; конструктор; графове представлення тексту; граф керування програми; алгоритм; відповідність алгоритму

Вступ

Невід’ємною частиною навчального процесу студентів спеціальності «Інженерія програмного забезпечення» є вивчення, побудова та реалізація алгоритмів. Важливо правильно інтерпретувати позначення, які використовують під час запису алгоритму, оскільки це дозволяє коректно його реалізувати у вигляді програмного коду. Студентам для самоконтролю, а викладачам для оцінювання якості виконаної роботи необхідна перевірка відповідності написаного програмного коду розробленому алгоритму.

Процес перевірки ускладнюється такими факторами як: складність алгоритму, що зумовлено великою кількістю складових (блоків) та

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп’ятник, Б. М. Яковенко, 2021

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського
національного університету залізничного транспорту, 2021, № 4 (94)

їх вкладеністю; довжина реалізації алгоритму; можлива зміна та доповнення позначень змінних; великі обсяги даних (для викладача) тощо.

Автоматизація цього процесу дозволить усім його учасникам заощадити час. Для її реалізації необхідна розробка та формалізація методу визначення відповідності тексту й алгоритму програми, що передбачає побудову відповідних математичних моделей обробки вхідних даних. Вхідними даними є текст (код) програми, написаний мовою високого рівня, та алгоритм, представлений у вигляді блок-схеми.

Також сферою застосування зазначеного методу є перевірка документації, яка містить представлення алгоритмів та програм, на академічний плагіат. Така перевірка потребує зіставлення алгоритмів між собою та аналогічне зіставлення текстів. Наявні наукові роботи присвячені розв'язанню задачі виявлення плагіату в природомовних текстах [7, 8, 9], у тому числі у структурованих документах [10], у програмному коді [4, 11, 6]. Також програмні коди зіставляють для визначення алгоритмічної схожості [5]. Верифікація програм дозволяє визначити відповідність програми її специфікації [1, 2]. Проте задача зіставлення й визначення відповідності програмного коду алгоритму є невирішеною.

Мета

Основною метою даної роботи є розробка методу визначення відповідності тексту (програмного коду) алгоритму програми на основі їх конструктивно-продукційних (КП) моделей. Метод передбачає виконання алгоритмів перетворення вхідних даних у графи керування та їх подальше зіставлення.

Методика

Для визначення відповідності тексту й алгоритму програми пропонується метод на основі КП-моделювання. Метод має такі складовікроки:

1. Попередня обробка тексту програми, що передбачає видалення незначимих частин коду (коментарі, пробіли тощо) та його розбиття на лексеми.
2. Проміжне представлення тексту програми, отриманого на попередньому кроці, у вигляді списку керуючих елементів (реалізують алгоритмічні структури вибору та циклу, а також операції передачі керування) та елементів процесу (відповідають алгоритмічній конструкції слідування).
3. Побудова графа керування програми за списком керуючих елементів.
4. Побудова графа керування за алгоритмом програми, що представлений блок-схемою.
5. Зіставлення графів керування програми та алгоритму шляхом обходу в ширину [3].

Для моделювання побудови графа керування програми та його проміжного представлення у вигляді списку застосуємо апарат конструктивно-продукційного моделювання [13, 14], в основі якого лежить поняття узагальненого конструктора:

$$C = \langle M, \Sigma, \Lambda \rangle, \tag{1}$$

де M – неоднорідний носій, який містить конструктивні елементи з атрибутами та може поповнюватися; Σ – сигнатура операцій (і відповідних відношень) зв'язування, підстановки й виведення, операцій над атрибутами; Λ – множина тверджень інформаційного забезпечення конструювання (ІЗК). ІЗК (конструктивна аксіоматика) містить онтологію, мету, правила, обмеження, початкові умови та умови завершення конструювання.

Призначення конструктора полягає у формуванні множин конструкцій за допомогою операцій сигнатури, які задають правилами ІЗК.

Для формування конструкцій необхідно виконувати ряд уточнюючих перетворень конструктора [13, 14]:

– спеціалізацію ($_S\mapsto$) – визначення предметної області застосування конструктора;

– інтерпретацію ($_I\mapsto$) – зв'язування операцій сигнатури з алгоритмами виконання деякої алгоритмічної структури;

– конкретизацію ($_K\mapsto$) – розширення ІЗК множиною правил продукцій, завдання конкретних множин нетермінальних і термінальних символів із їх атрибутами і, за необхідності, значень атрибутів;

– реалізацію ($_R\mapsto$) – послідовне виконання операції виведення для побудови конструкцій.

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

Результати

Конструктор списку. Спеціалізований конструктор для перетворення програмного коду в проміжне представлення у вигляді списку має вигляд:

$$C = \langle M, \Sigma, \Lambda \rangle \ {}_{S}\!\mapsto C_T = \langle M_T, \Sigma_T, \Lambda_T \rangle, \quad (2)$$

де M_T – неоднорідний розширюваний носій, що складається з термінальних і нетермінальних елементів; Σ_T – множина операцій і відношень на елементах M_T; Λ_T – ІЗК, що містить онтологію, мету, обмеження, правила, умови початку й завершення конструювання.

Онтологія конструктора C_T: розширюваний носій складається з множини термінальних і нетермінальних елементів $M_T = \{T_T \cup N_T\}$. Терміналами є $T_T = \{txt, list, \{P_i\}, \{O_i\}, \{Ob_i\}, \{L_i\}, \varepsilon\}$, де $txt \supset \{Ob_i\} \cup \{O_i\} \cup \{P_i\}$ – послідовність операторів програмного коду; $list$ – конструкція, що містить послідовність операторів програмного коду та допоміжних лексем; P_i – позначення процесу, що відповідає будь-якій послідовності операторів, що реалізує алгоритмічну структуру «слідування»; $\{O_i\} = \{return,\ break, continue\}$ – множина керуючих операторів без тіла; $\{Ob_i\} = \{if, else, do, while, for, switch, case, default\}$ – множина керуючих операторів, що можуть мати власне тіло; $\{L_i\} = \{"++", "--"\}$ – множина допоміжних лексем для заповнення списку, ε – порожній елемент (символ).

Розглянемо сигнатуру:

$$\Sigma_T = \langle \Theta_T, \{\rightarrow\}, \{+\} \rangle \cup \Psi_G, \quad (3)$$

де $\Theta_T = \{\Rightarrow, |\Rightarrow, ||\breve{\Rightarrow}\}$ – множина операцій виведення; $\rightarrow$ – відношення підстановки; Ψ_G – множина правил продукції виведення $\psi_i = \langle \hat{s}_i, \breve{s}_i \rangle$, де i – номер правила, $\hat{s}_i$ – правило підстановки для розпізнавання тексту програмного коду, $\breve{s}_i$ – правило підстановки для побудови стеку; $+(el, list)$ – операція додавання нової вершини до списку, де el – значення нової вершини, $list$ – список.

Метою конструювання є побудова проміжного представлення програмного коду у вигляді списку.

Обмеження конструктора C_T накладають конструкцією програмного коду.

Початкова умова конструювання: η – нетермінал, із якого починається побудова конструкції списку; α – нетермінал, із якого починається розпізнавання тексту програми.

Умова завершення конструювання: форма не містить нетерміналів, побудована конструкція списку відповідає програмному коду.

У результаті конкретизації конструктора $C_T \ {}_{K}\!\mapsto {}_{K} C_T$ маємо такі правила підстановки:

$$\hat{s}_0 = \langle \alpha \rightarrow \gamma\alpha \,|\, \beta\alpha \rangle; \quad (4)$$

$$\hat{s}_1 = \langle \beta \rightarrow P_1 \,|\, P_2 \ldots P_n \rangle, \quad (5)$$

$$\breve{s}_1 = \langle \eta \rightarrow +(p, list)\eta \rangle;$$

$$\hat{s}_2 = \hat{s}_3 = \langle \gamma \rightarrow C_1 \,|\, C_2 \ldots C_n \rangle, \quad (6)$$

$$\breve{s}_2 = \langle \eta \rightarrow +(Ob_i, list) + +("++", list)\eta + ("--", list)\eta \rangle;$$

$$\breve{s}_3 = \langle \eta \rightarrow +(O_i, list)\eta \rangle; \quad (7)$$

$$\breve{s}_4 = \langle \eta \rightarrow \varepsilon \rangle. \quad (8)$$

У ході інтерпретації проведемо зв’язування операцій сигнатури Σ_T з алгоритмами їх виконання:

$$\langle C_T = \langle M_T, \Sigma_T, \Lambda_T \rangle, C_A = \langle M_A, \Sigma_A, \Lambda_A \rangle \rangle \ {}_{I}\!\mapsto {}_{I}\!\mapsto_{I, C_A} C_T = \langle M_T, \Sigma_T, \Lambda_1 \rangle, \quad (9)$$

де $M_A \supset V_A, V_A = \{A_i^0 |_{X_i}^{Y_i}\}$ – множина базових алгоритмів; X_i, Y_i – множини визначення та значень алгоритму $A_i^0 |_{X_i}^{Y_i}$;

$\Lambda_A = \left\{ M_A = \bigcup_{A_i^o \in V_A} \left(X\left(A_i^0\right) \subset Y\left(A_i^0\right) \right) \cup \Omega(C_T) \right\}$ –

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп’ятник, Б. М. Яковенко, 2021

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)
Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 4 (94)

неоднорідний носій; $\Omega(C_T)$ – множина конструкцій списків, які задовольняють C_T; $\Lambda_1 = \Lambda_T \bigcup \Lambda_A \bigcup \Lambda_2$.

Конструктор ${}_{I,C_A}C_T$ містить алгоритми виконання операцій: $\Lambda_2 = \left\{ \left(A_1^0 \left|_{A_i, A_j}^{A_i \cdot A_j} \right. \dashv \cdot \right), \right.$ $\left(A_2^0 \left|_{S}^{A_1} \right. \dashv : \right)$, $\left(A_3 \left|_{el, list}^{list} \right. \dashv + \right)$, $\left(A_4 \left|_{l_h, l_q, f_i}^{f_i} \right. \dashv \Rightarrow \right)$, $\left. \left(A_5 \left|_{f_i, \Psi}^{f_j} \right. \dashv |\Rightarrow \right), \left(A_6 \left|_{\sigma, \Psi}^{\bar{\Omega}} \right. \dashv \| \Rightarrow \right) \right\}$.

Результатом реалізації конструктора (9) є множина конструкцій списків, які є проміжним представлення тексту програми та відповідного йому графа керування.

Конструктор графа. Спеціалізований конструктор для побудови графа керування програми за її проміжним представлення у вигляді списку керуючих операторів буде таким:

$$C = \langle M, \Sigma, \Lambda \rangle \ {}_S{\mapsto} \ C_G = \langle M_G, \Sigma_G, \Lambda_G \rangle, \quad (10)$$

де M_G – неоднорідний розширюваний носій; Σ_G – множина операцій і відношень на елементах M_G; Λ_G – ІЗК.

Онтологія конструктора C_G: розширюваний носій складається з множини термінальних і нетермінальних елементів $M_G = \{T_G \bigcup N_G\}$. Терміналами є конструкції графів та їх складові, проміжне представлення програмних кодів у вигляді списків та допоміжних конструкцій:

$$T_G = \{G \bigcup V \bigcup E \bigcup list \bigcup stack_temp \bigcup \quad (11)$$
$$\bigcup stack_root \bigcup stack_break \bigcup stack_next$$
$$\bigcup stack_continue \bigcup stack_switch \bigcup$$
$$\bigcup temp \bigcup l_vertex\},$$

де $G = \langle V, E \rangle$ – конструкція графа; $V = \{v_i\}$, $E = \{e_i\}$ – множина вершин і дуг відповідно; $list$ – побудована конструкція C_T; допоміжні конструкції-стеки вершин графа з навантаженням: у вигляді назв керуючих операторів – $stack_temp$, які є коренем підграфа, що утворюється під час роботи з операторами $\{if, else, while, for, case\}$ – $stack_root$; "$break$" – $stack_break$; "$continue$" – $stack_continue$; які мають спільні дуги з вершиною, що буде додана – $stack_next$; "$case$" – $stack_switch$; $temp$ зберігає вершину допоміжних стеків, l_vertex зберігає вершину $list$. Допоміжні конструкції стеків є впорядкованими наборами елементів, що формуються за принципом LIFO.

Вершина графа має атрибути $w_v \dashv v = \langle index, name \rangle$, де $index$ – індекс вершини, набуває цілочислових значень, $name$ – навантаження вершини. Атрибути дуги $w_e \dashv e = \langle start, finish \rangle$, де $start, finish$ – цілочислові індекси інцендентних вершин.

Граф має атрибути $w_G = \langle begin, end, current_v, current_e \rangle$, де $begin$ – початкова вершина графу, end – кінцева вершина графа, $current_v = \langle name, index, prev \rangle$ – поточна вершина під час формування графа, де $prev$ – попередня вершина графа, $current_e = \langle start, finish \rangle$ – поточна дуга під час формування графа.

Атрибути вершини проміжного представлення програмного коду l_vertex – $w_l = \langle name, prev \rangle$, де $name$ – навантаження вершини списку, $prev$ – навантаження попередньої вершини списку.

Атрибути вершини допоміжного стеку $temp$ – $w_{ST} = w_v$.

Розглянемо сигнатуру Σ_G:

$$\Sigma_G = \langle \Xi_G, \Theta_G, \Phi_G, \{\to\} \rangle \bigcup \Psi_G, \quad (11)$$

де $\Xi_G = \left\{ \tilde{\bigcup}, \bigcup \right\}$ – множина операцій зв'язування; $\Theta_G = \{\Rightarrow, |\Rightarrow, \|\breve{\Rightarrow}\}$ – множина операцій виведення; $\Phi_G = \{\div, :=, \%, \#, +, -, —, \times\}$ – множина операцій над атрибутами; $\to$ – відношення підстановки.

Розглянемо операції над атрибутами:

– $\div(c, n, L)$ – виконання n операцій зі списку L, якщо $c = true$;

$a := b$ – присвоєння, копіює значення операнду b в a;

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

– $\%(index, V)$ – знаходження навантаження вершини, індекс якої дорівняє $index$ у множині вершин V;

– $\#Q$ – обчислення потужності множини, визначає число, яке дорівнює кількості елементів у Q;

– $+(el, stack)$ – додавання елемента el до допоміжного стеку $stack$;

– $-(el, stack)$ – вилучення вершини допоміжного стеку $stack$ в el за умови, що $stack$ не порожній;

– $--(el, list)$ – вилучення вершини списку $list$ та збереженні її в el за умови, що $list$ не порожній;

– $\times(a,b)$ – множення двох чисел, передбачає знаходження третього числа, що є їхнім добутком.

Операція об'єднання графів $w_G \lrcorner G = \tilde{\bigcup}(w_1 \lrcorner G_1, w2 \lrcorner G_2)$ передбачає формування нового графа $w_G \lrcorner G$, що містить об'єднані множини вершин і дуг вихідних графів $w_G \lrcorner G = \langle V, E \rangle$, де $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, $w_1 \lrcorner G_1 = \langle V_1, E_1 \rangle$, $w_2 \lrcorner G_2 = \langle V_2, E_2 \rangle$, при цьому $\cup$ – традиційна операція об'єднання.

Відношення підстановки має вигляд:

$$\psi_i = \langle s_i, g_i \rangle,\ s_i = \langle \bar{s}_i, \tilde{s}_i \rangle,\ g_i = \langle \bar{g}_i, \tilde{g}_i \rangle, \quad (12)$$

де $\bar{s}_i, \tilde{s}_i$ – відношення підстановки для роботи зі списком і побудови конструкції графа відповідно; $\bar{g}_i, \tilde{g}_i$ – операції над атрибутами списку та графа, його вершин і дуг відповідно. Якщо операцію над атрибутами не виконують, відношення підстановки має вигляд $\psi_i = \langle s_i, \varepsilon \rangle$.

Операція повного виведення $\|\tilde{\Rightarrow}(\Psi, w_l \lrcorner l)$ та більш детальна інформація щодо інших операцій наведена в роботах [13, 14]. Результатом операції виведення є конструкції-граф.

Метою конструювання є побудова графа керування програми за проміжним представленням у вигляді $list \in \Omega(C_T)$.

Обмеження конструктора C_G накладають навантаженням вершин у списку.

Початкова умова конструювання: σ – нетермінал, із якого починається побудова конструкції графа керування; ρ – нетермінал, із якого починається вилучення вершин із $list \lrcorner C_T$.

Умова завершення конструювання: форма не містить нетерміналів, конструкція списку порожня.

У результаті конкретизації конструктора $C_G\ {}_K{\mapsto}\ {}_K C_G$ маємо нижченаведені правила підстановки.

Правило ініціалізації графа має вигляд:

$$\bar{s}_1 = \varepsilon,\ \bar{g}_1 = \varepsilon; \quad (13)$$

$$\tilde{s}_1 = \langle \sigma \to G\alpha \rangle;$$

$$\tilde{g}_1 = \langle index \lrcorner v_1 := 1, name \lrcorner v_1 := "begin",$$

$$index \lrcorner v_2 := 0, name \lrcorner v_2 := "end", begin \lrcorner G := v_1,$$

$$end \lrcorner G := v_2, current \lrcorner G := v_1, V = \{v_1, v_2\} \rangle.$$

Побудова графа передбачає вилучення вершини конструкції $list$ ($\bar{s}_2$), тому далі для правил $G\alpha$ та $G\beta$, якщо $\bar{s}_i$ або $\bar{g}_i$ не вказано, відповідно $\bar{s}_i = \bar{s}_2$, $\bar{g}_i = \bar{g}_2$, а для правил Gf, $G\beta$ та $G\sigma$, якщо $\bar{s}_i$ або $\bar{g}_i$ не вказано, $\bar{s}_i = \varepsilon$, $\bar{g}_i = \varepsilon$:

$$\bar{s}_2 = \langle \rho \to --(l_vertex, list) \rangle; \quad (14)$$

$$\bar{g}_2 = \varepsilon.$$

Правило для додавання нових вершин до графа має вигляд:

$$\tilde{s}_2 = \left\langle G\alpha\ {}_{d_1} \to \tilde{\bigcup}(G, G^*)\alpha \right\rangle; \quad (15)$$

$$\tilde{g}_2 = \langle \div(name \lrcorner l_vertex = P \mid el \in \{Cb_i\} \setminus$$

$$\setminus \{"do", "else"\}, 9, d_1 := true, G^* = V^* \cup E^*,$$

$$V^* = \{v\}, E^* = \{e\}, name \lrcorner v := name \lrcorner l_vertex,$$

$$index \lrcorner v := \#G, start \lrcorner e := index \lrcorner current \lrcorner G,$$

$$finish \lrcorner e := index \lrcorner v, current \lrcorner G := v) \rangle.$$

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

Далі наведемо правила для додавання вершин до конструкції $stack_temp$:

– вершина "do":

$$\tilde{s}_3 = \left\langle G\alpha \;_{d_2}\!\rightarrow G\alpha \right\rangle; \tag{16}$$

$$\tilde{g}_3 = \Big\langle \div\Big(name\hookleftarrow l_vertex = "++" \& \, name\hookleftarrow prev\hookleftarrow$$

$$\hookleftarrow l_vertex = "do", 2, d_2 := true,$$

$$+\big(-1 \times (\#G + 1), stack_temp\big)\Big\rangle.$$

– вершина "$else$":

$$\tilde{s}_4 = \left\langle G\alpha \;_{d_3}\!\rightarrow G\alpha \right\rangle; \tag{17}$$

$$\tilde{g}_4 = \Big\langle \div\Big(name\hookleftarrow l_vertex = "++" \& \, name\hookleftarrow prev\hookleftarrow$$

$$\hookleftarrow l_vertex = "else", 2, d_3 := true,$$

$$+\big(-1, stack_temp\big)\Big\rangle;$$

– інші вершини:

$$\tilde{s}_5 = \left\langle G\alpha \;_{d_4}\!\rightarrow G\alpha \right\rangle; \tag{18}$$

$$\tilde{g}_5 = \Big\langle \div\Big(name\hookleftarrow l_vertex = "++" \& \, name\hookleftarrow prev\hookleftarrow$$

$$\hookleftarrow l_vertex != el \in \{"do", "else"\}, 2, d_4 := true,$$

$$+\big(index\hookleftarrow current\hookleftarrow G, stack_temp\big)\Big\rangle.$$

Правила для обробки різних вершин конструкцій мають вигляд:

– "$return$" конструкції $list$:

$$\tilde{s}_6 = \left\langle G\alpha \;_{d_5}\!\rightarrow \tilde{\bigcup}\left(G, G^*\right)\alpha \right\rangle; \tag{19}$$

$$\tilde{g}_6 = \Big\langle \div\Big(name\hookleftarrow l_vertex = "return", 5, d_5 := true,$$

$$G^* = E^*, E^* = \{e\}, start\hookleftarrow e_i := index\hookleftarrow current\hookleftarrow G,$$

$$finish\hookleftarrow e_i := index\hookleftarrow end\hookleftarrow G \Big\rangle;$$

– "$break$" конструкції $list$:

$$\tilde{s}_7 = \left\langle G\alpha \;_{d_6}\!\rightarrow G\alpha \right\rangle; \tag{20}$$

$$\tilde{g}_7 = \Big\langle \div\Big(name\hookleftarrow l_vertex = "break", 2, d_6 := true,$$

$$+\big(index\hookleftarrow current\hookleftarrow G, stack_break\big)\Big\rangle;$$

– "$continue$" конструкції $list$:

$$\tilde{s}_8 = \left\langle G\alpha \;_{d_7}\!\rightarrow G\alpha \right\rangle; \tag{21}$$

$$\tilde{g}_8 = \Big\langle \div\Big(name\hookleftarrow l_vertex = "continue", 2,$$

$$d_7 := true, +\big(index\hookleftarrow current\hookleftarrow G, stack_continue\big)\Big\rangle;$$

– "$--$" конструкції $list$:

$$\tilde{s}_9 = \left\langle G\alpha \;_{d_8}\!\rightarrow G\phi\alpha \right\rangle; \tag{22}$$

$$\tilde{g}_9 = \Big\langle \div\Big(name\hookleftarrow l_vertex = "--", 1, d_8 := true\Big)\Big\rangle;$$

– "$else$" конструкції $stack_temp$:

$$\tilde{s}_{10} = \left\langle G\phi \;_{d_9}\!\rightarrow G\phi \right\rangle; \tag{23}$$

$$\tilde{g}_{10} = \Big\langle -(temp, stack_temp), \div\Big(temp = -1, 2,$$

$$d_8 := true, +\big(index\hookleftarrow current\hookleftarrow G, stack_root\big)\Big)\Big\rangle;$$

– "do" конструкції $stack_temp$:

$$\tilde{s}_{11} = \left\langle G\phi \;_{d_{10}}\!\rightarrow \tilde{\bigcup}\left(G\beta, G^*\right)\chi\phi \right\rangle; \tag{24}$$

$$\tilde{g}_{11} = \Big\langle -(temp, stack_temp), \div\Big(temp < -1, 5,$$

$$d_{10} := true, G^* = E^*, E^* = \{e\},$$

$$start\hookleftarrow e_i := index\hookleftarrow current\hookleftarrow G,$$

$$finish\hookleftarrow e_i := temp \times -1\Big)\Big\rangle.$$

Правило для додавання нової вершини до графа має вигляду:

$$\tilde{s}_{12} = \left\langle G\beta \rightarrow \tilde{\bigcup}\left(G, G^*\right) \right\rangle; \tag{25}$$

$$\tilde{g}_{12} = \Big\langle G^* = V^* \cup E^*, V^* = \{v\}, E^* = \{e\},$$

$$name\hookleftarrow v := name\hookleftarrow l_vertex, index\hookleftarrow v := \#G,$$

$$start\hookleftarrow e := index\hookleftarrow current\hookleftarrow G,$$

$$finish\hookleftarrow e := index\hookleftarrow v, current\hookleftarrow G := v\Big\rangle.$$

Creative Commons Attribution 4.0 International

doi: https://doi.org/10.15802/stp2021/245666

© О. С. Куроп'ятник, Б. М. Яковенко, 2021

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 4 (94)

Правило для обробки вершин "*while*" та "*for*" конструкції $stack_temp$:

$$\tilde{s}_{13} = \left\langle G\phi_{\ d_{11}} \to \tilde{\bigcup}\left(G, G^*\right)\chi\phi \right\rangle; \quad (26)$$

$$\tilde{g}_{13} = \Big\langle -(temp, stack_temp), \div\Big(\%\left(temp, V\right) =$$

$$= "while" | "for", 6, d_{11} := true, G^* = E^*, E^* = \{e\},$$

$$start \hookleftarrow e_i := index \hookleftarrow current \hookleftarrow G, finish \hookleftarrow e_i := temp,$$

$$+(temp, stack_root)\Big)\Big\rangle .$$

Далі подано правила для обробки вершин конструкцій $stack_root$ та $stack_continue$ у правилі $\tilde{s}_{13}$:

$$\tilde{s}_{14} = \left\langle G\chi \to \tilde{\bigcup}\left(G, G^*\right)\chi \right\rangle; \quad (27)$$

$$\tilde{g}_{14} = \Big\langle G^* = E^*, E^* = \{e\}, -(temp, stack_root),$$

$$start \hookleftarrow e := temp, finish \hookleftarrow e := finish \hookleftarrow current_e \hookleftarrow G \Big\rangle;$$

$$\tilde{s}_{15} = \left\langle G\chi \to \tilde{\bigcup}\left(G, G^*\right)\chi \right\rangle; \quad (28)$$

$$\tilde{g}_{15} = \Big\langle G^* = E^*, E^* = \{e\},$$

$$-(temp, stack_continue), start \hookleftarrow e := temp,$$

$$finish \hookleftarrow e := finish \hookleftarrow current_e \hookleftarrow G \mid$$

$$\mid start \hookleftarrow current_e \hookleftarrow G \Big\rangle;$$

$$\tilde{s}_{16} = \langle \chi \to \varepsilon \rangle . \quad (29)$$

Правила для обробки вершини "*if*" конструкції $stack_temp$:

$$\tilde{s}_{17} = \left\langle G\phi_{\ d_{12}} \to G\phi \right\rangle; \quad (30)$$

$$\tilde{g}_{17} = \Big\langle -(temp, stack_temp), \div\Big(\%\left(temp, V\right) = "if",$$

$$3, d_{12} := true, +(temp, stack_next),$$

$$+(index \hookleftarrow current \hookleftarrow G, stack_root)\Big)\Big\rangle ;$$

$$\tilde{s}_{18} = \left\langle G\phi_{\ d_{13}} \to G\phi \right\rangle; \quad (31)$$

$$\tilde{g}_{18} = \Big\langle -(temp, stack_temp), \div\Big(\%\left(temp, V\right) = "if",$$

$$3, d_{13} := true, +(temp, stack_root),$$

$$+(index \hookleftarrow current \hookleftarrow G, stack_root)\Big)\Big\rangle .$$

Подамо правило для обробки вершини "*case*" та "*default*" конструкції $stack_temp$:

$$\tilde{s}_{19} = \left\langle G\phi_{\ d_{14}} \to G\phi \right\rangle; \quad (32)$$

$$\tilde{g}_{19} = \Big\langle -(temp, stack_temp), \div\Big(\%\left(temp, V\right) =$$

$$= "case" | "default", 2, d_{14} := true,$$

$$+(temp, stack_switch)\Big)\Big\rangle .$$

Правило для обробки вершини "*switch*" конструкції $stack_temp$ має вигляд:

$$\tilde{s}_{19} = \left\langle G\phi_{\ d_{15}} \to G\delta\phi \right\rangle; \quad (33)$$

$$\tilde{g}_{19} = \Big\langle -(temp, stack_temp), \div\Big(\%\left(temp, V\right) =$$

$$= "switch", 1, d_{15} := true\Big)\Big\rangle .$$

Правила для обробки вершин конструкцій $stack_switch$:

$$\tilde{s}_{20} = \left\langle G\delta \to \tilde{\bigcup}\left(G, G^*\right)\delta \right\rangle; \quad (34)$$

$$\tilde{g}_{20} = \Big\langle G^* = E^*, E^* = \{e\}, -(temp, stack_switch),$$

$$start \hookleftarrow e := start \hookleftarrow current_e \hookleftarrow G, finish \hookleftarrow e := temp\Big)\Big\rangle ;$$

$$\tilde{s}_{21} = \langle \delta \to \varepsilon \rangle . \quad (35)$$

Правила для обробки вершини конструкції $stack_break$:

$$\tilde{s}_{22} = \langle G\phi \to G\phi \rangle; \quad (36)$$

$$\tilde{g}_{22} = \Big\langle -(temp, stack_break),$$

$$+(temp, stack_next)\Big\rangle ;$$

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

$$\tilde{s}_{23} = \langle \phi \to \varepsilon \rangle . \qquad (37)$$

Правило для обробки вершини конструкції $stack_next$ має вигляд:

$$\tilde{s}_{24} = \left\langle G\alpha \to \tilde{\bigcup}\left(G, G^{*}\right)\alpha \right\rangle ; \qquad (38)$$

$$\tilde{g}_{24} = \langle G^{*} = E^{*}, E^{*} = \{e\}, -(temp, stack_next),$$

$$start \lrcorner e := temp, finish \lrcorner e := index \lrcorner current_{v} \lrcorner G \rangle .$$

Правило для обробки вершини конструкції $stack_root$:

$$\tilde{s}_{25} = \left\langle G\alpha \to \tilde{\bigcup}\left(G, G^{*}\right)\alpha \right\rangle ; \qquad (39)$$

$$\tilde{g}_{25} = \langle G^{*} = E^{*}, E^{*} = \{e\}, -(temp, stack_root),$$

$$start \lrcorner e := temp, finish \lrcorner e := index \lrcorner current_{v} \lrcorner G \, | .$$

$$| \, index \lrcorner end \lrcorner G \rangle .$$

Правила для додавання дуги до останньої вершини графа:

$$\tilde{s}_{26} = \left\langle G\alpha \to \tilde{\bigcup}\left(G, G^{*}\right)\alpha \right\rangle ; \qquad (40)$$

$$\tilde{g}_{26} = \langle G^{*} = E^{*}, E^{*} = \{e\},$$

$$start \lrcorner e := index \lrcorner current_{v} \lrcorner G,$$

$$finish \lrcorner e := index \lrcorner end \lrcorner G \rangle .$$

Наступне правило дозволяє завершити обробку вершин конструкції $stack$:

$$\tilde{s}_{27} = \langle \alpha \to \varepsilon \rangle . \qquad (41)$$

У ході інтерпретації проведено зв’язування операцій сигнатури Σ_G з алгоритмами їх виконання:

$$\left\langle C_G = \langle M_G, \Sigma_G, \Lambda_G \rangle, C_{A'} = \langle M_{A'}, \Sigma_{A'}, \Lambda_{A'} \rangle \right\rangle {}_{I}\!\mapsto$$

$${}_{I}\!\mapsto_{I, C_{A'}} C_G = \langle M_G, \Sigma_G, \Lambda_3 \rangle , \qquad (42),$$

де $M_{A'} \supset V_{A'}, V_{A'} = \left\{ A_i^0 |_{X_i}^{Y_i} \right\}$ – множина базових алгоритмів; X_i, Y_i – множини визначення та значень алгоритму $A_i^0 |_{X_i}^{Y_i}$;

$$\Lambda_{A'} = \left\{ M_{A'} = \bigcup_{A_i^o \in V_A} \left(X\left(A_i^0\right) \subset Y\left(A_i^0\right) \right) \bigcup \Omega\left(C_G\right) \right\} -$$

неоднорідний носій; $\Omega\left(C_G\right)$ – множина конструкцій графів, які задовольняють C_G; $\Lambda_3 = \Lambda_G \bigcup \Lambda_{A'} \bigcup \Lambda_4$.

Конструктор ${}_{I, C_A} C_G$ містить алгоритми виконання операцій: $\Lambda_4 = \Big\{ \left(A_1^0 |_{A_i, A_j}^{A_i \cdot A_j} \lrcorner \cdot \right)$, $\left(A_2^0 | _{S}^{A_1} \lrcorner : \right)$, $\left(A_3 |_{l_h, l_q, f_i}^{f_i} \lrcorner \Rightarrow \right)$, $\left(A_4 |_{f_i, \Psi}^{f_j} \lrcorner | \Rightarrow \right)$, $\left(A_5 |_{\sigma, \Psi}^{\bar{\Omega}} \lrcorner || \Rightarrow \right)$, $\left(A_6 |_{c,n,L}^{L} \lrcorner \div \right)$, $\left(A_7 |_{a,b}^{a} \lrcorner := \right)$, $\left(A_8 |_{index, V}^{name} \lrcorner \% \right)$, $\left(A_9 |_{Q}^{x} \lrcorner \# \right)$, $\left(A_{10} |_{el, stack}^{stack} \lrcorner + \right)$, $\left(A_{11} |_{el, stack}^{el} \lrcorner - \right)$, $\left(A_{12} |_{el, list}^{el} \lrcorner -- \right)$, $\left(A_{13} |_{a,b}^{c} \lrcorner \times \right)$, $\left(A_{14} |_{Q_1, Q_2}^{Q} \lrcorner \bigcup \right)$, $\left(A_{15} |_{G_1, G_2}^{G} \lrcorner \tilde{\bigcup} \right) \Big\}$.

Результатом реалізації конструктора (42) є множина конструкцій графів керування, побудованими за конструкціями з $\Omega\left(C_T\right)$.

Таким чином, правила (13) – (41) дозволяють перетворити текст програми, попередньо побудований за правилами (4) – (8), у граф керування. При цьому враховані оператори керування $\{C_i\} \bigcup \{Cb_i\}$. Інші оператори мають уніфіковане представлення у вигляді вершини-процесу.

Приклад використання моделі. Розглянемо приклад побудови графа керування для функції знаходження мінімуму з двох чисел, написаної мовою програмування C++. Програмний код функції:

```
int min(int a, int b)
{
   int _min = a;
   if(_min>b) _min = b;
   return _min;
}.
```

Наведений код було перетворено в $list$-конструкцію, побудовану за допомогою C_T (рис. 1, *a*). Head – голова списку, принцип робо-

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп’ятник, Б. М. Яковенко, 2021

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського
національного університету залізничного транспорту, 2021, № 4 (94)

ти списку – FIFO. Граф потоку керування (рис. 1, *б*) будуємо за допомогою послідовного виконання правил:

$$\sigma\ {}_1\rightarrow G\alpha\ {}_2\rightarrow \tilde{\cup}\left(G,G^*\right)\alpha\ {}_2\rightarrow, \qquad (44)$$

$${}_2\rightarrow \tilde{\cup}\left(G,G^*\right)\alpha\ {}_5\rightarrow G\alpha\ {}_2\rightarrow \tilde{\cup}\left(G,G^*\right)\alpha\ {}_9\rightarrow$$

$${}_9\rightarrow G\phi\alpha\ {}_{18}\rightarrow G\phi\ {}_{23}\rightarrow G\alpha\ {}_{25}\rightarrow$$

$${}_{25}\tilde{\rightarrow}\cup\left(G,G^*\right)\ {}_{25}\rightarrow \tilde{\cup}\left(G,G^*\right)\ {}_{27}\rightarrow$$

$${}_{27}\rightarrow \varepsilon\,,$$

де числові індекси стрілки є i-індексами правила $\tilde{S}_i$.

На рис. 1, *б* вершини графа містять позначки вершини списку, яким відповідають, а також номери вершини, що зазначено в дужках.

Об'єктно-орієнтована модель графа керування програми. Для зіставлення алгоритмів, представлених у вигляді блок-схем, та текстів програм подамо їх як графи керування. Граф керування будуємо за блок-схемою алгоритму, створеною в онлайн-редакторі [12]. Цей редактор дозволяє отримати схему у форматі json, розбір якого відбувається в такому порядку:

1. побудова проміжної моделі блок-схеми, що містить два класи, які відповідають за зчитування даних із файлу .json;
2. побудова графа керування на основі моделі блок-схеми, побудованої на попередньому кроці.

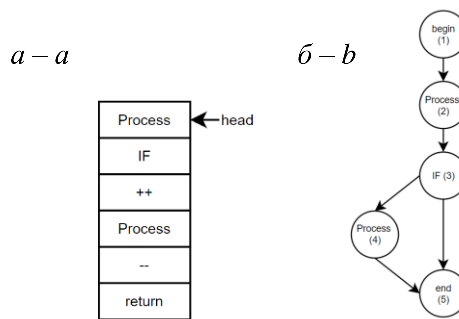

Рис. 1. Схема конструкцій, побудованих C_T і C_G: *а* – список керуючих операторів; *б* – граф керування

Fig. 1. Scheme of structures constructed by C_T and C_G: *a* – list of control operators; *b* – control graph

Граф керування програми будуємо за моделлю, описаною в (2) – (41).

Для програмної реалізації моделі графа керування програми виконаємо її об'єктно-орієнтоване (ОО) моделювання з використанням UML. Реалізацію моделі покладено на класи: PreprocessingCode, LexicalAnalysis, Graph (рис. 2).

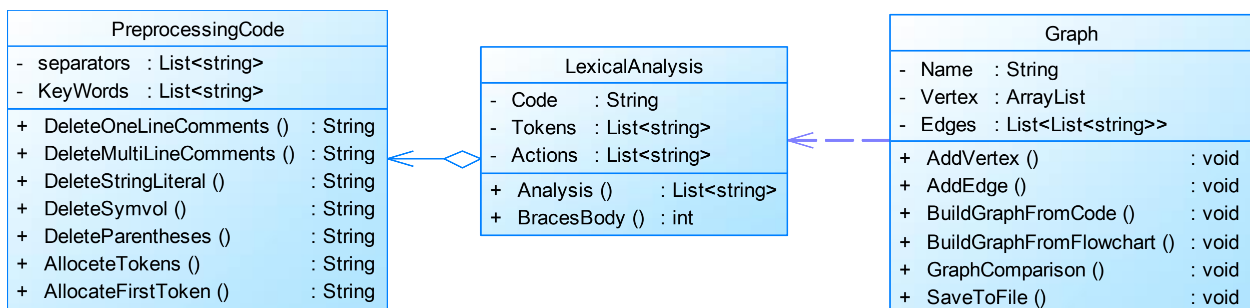

Рис. 2. Класове представлення конструкторів списку та графа керування

Fig. 2. Class representation of list and control graph constructors

Розглянемо відповідність алгоритмів конструктора стеку (9) методам класів (рис. 2):

A_1^0 – композиция алгоритмів – присутній у всіх методах, оскільки кожен із них складається з декількох підалгоритмів, таких як присвоєння, порівняння, об'єднання та інші;

A_2^0 – умовне виконання алгоритму – присутній у всіх методах, оскільки кожен із них містить оператор умовного виконання, що визначає використання тих чи інших алгоритмів;

A_3 – додавання вершини до списку – реалізовано методом LexicalAnalysis::BracesBody();

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського
національного університету залізничного транспорту, 2021, № 4 (94)

$A_4 - A_6$ – реалізовано методом LexicalAnalysis::BracesBody(), вони дозволяють послідовно заповнити стек.

Розглянемо відповідність алгоритмів конструктора графа (42) методам класів (рис. 1):

A_1^0, A_2^0 – відповідають алгоритмам, описаним у конструкторі списку;

$A_3 - A_5$ – реалізовано методом Graph::BuildGraphFromCode(), вони дозволяють послідовно побудувати граф;

A_6 – умовне виконання списку операцій – присутній у всіх методах;

A_7 – присвоєння операндів – присутній у всіх методах;

A_8 – пошук вершини графа за індексом – реалізовано у методі Graph::BuildGraphFromCode();

A_9 – обчислення потужності множини – реалізовано у вигляді параметрів атрибутів класів;

A_{10} – додавання елемента до допоміжного стеку – реалізовано у методі Graph::BuildGraphFromCode();

A_{11} – вилучення вершини допоміжного стеку – реалізовано у методі Graph::BuildGraphFromCode();

A_{12} – вилучення вершини списку конструкції, побудованої конструктором C_T, – реалізовано у методі Graph::BuildGraphFromCode();

A_{13} – множення двох чисел – реалізовано у методі Graph::BuildGraphFromCode();

A_{14}, A_{15} – об'єднання множин та графів відповідно – реалізовано у методі Graph::BuildGraphFromCode().

Порівняння графів, що відповідають тексту й алгоритму програми, виконуємо на основі методу пошуку в ширину: алгоритм перераховує всі досяжні з S вершини в порядку зростання відстані від S [1]. Перегляд вершин є паралельним на обох графах, а вершини помічають, якщо вони збігаються в обох графах.

Далі схожість графів виражаємо в числовому еквіваленті за формулою:

$$Match(A,B) = \frac{|A'| + |B'| - 2}{|A| + |B| - 2} \times 100, \qquad (43)$$

де A, B – досліджувані графи; $|A|$, $|B|$ – кількість вершин у відповідних графах; $|A'|$, $|B'|$ – кількість помічених верши. За порівняння графів відповідає метод Graph::GraphComparison().

Для використання побудованих графових моделей та повної реалізації методу визначення відповідності тексту й алгоритму програми було розроблено та реалізовано ОО-модель додатку з GUI (рис. 3).

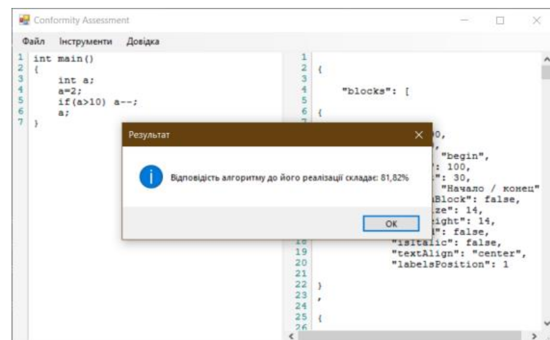

Рис. 3. Вікно додатку для зіставлення тексту й алгоритму програми

Fig. 3. Application window for comparing the text and algorithm of the program

Таким чином, метод для визначення відповідності програмного коду алгоритму, який він реалізує, передбачає використання розробленої системи конструкторів для перетворення програмного коду в граф керування. На її основі побудована об'єктно-орієнтована модель для конструювання графа керування програми та графа керування алгоритму.

Модель реалізовує побудову графів програми, алгоритму та їх порівняння, тобто повністю виконує кроки запропонованого методу.

Наукова новизна та практична значимість

У роботі отримали подальший розвиток методи конструктивно-продукційного моделювання в задачах обробки текстів, написаних штучними мовами. Запропоновано метод зіставлення алгоритму та його програмної реалізації. Метод передбачає використання розробленої системи конструкторів, що виконує перетворення тексту програм мовою C++ у граф керування для подальшого зіставлення з алгоритмом. Отримані ре-

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

зультати мають значення для розв'язання таких задач, як зіставлення текстів програм із метою виявлення запозичень, визначення відповідності алгоритмів програм їх програмним реалізаціям із метою поліпшення навичок кодування. Графове представлення, яке продукує розроблена система конструкторів, може бути застосоване для дослідження впливу оптимізації та рефакторингу коду на складність програм із використанням метрик МакКейба.

Висновки

Запропоновано метод визначення відповідності тексту алгоритму програми. Використання апарату конструктивно-продукційного моделювання дозволило формалізувати процеси побудови графа керування програми, при цьому можливість розширення множини правил інформаційного забезпечення конструктора дає можливість в майбутньому врахувати класову структуру програми та модульність у цілому. Наявність поповнюваного носія в конструкторі дозволяє розширити модель для інших мов програмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ивутин А. Н., Трошина А. Н. Метод формальной верификации параллельных программ с использованием сетей Петри. *Вестник Рязанского государственного радиотехнического университета*. 2019. № 70. С. 15–26. DOI: https://doi.org/10.21667/1995-4565-2019-70-15-26
2. Кондратьев Д. А., Марьясов И. В., Непомнящий В. А. Автоматизация верификации С-программ с использованием символического метода элиминации инвариантов циклов. *Моделирование и анализ информационных систем.* 2018. № 25 (5). С. 491–505. DOI: https://doi.org/10.18255/1818-1015-2018-5-491-505
3. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. *Алгоритмы. Построение и анализ.* Москва : Вильямс, 2005. 1296 с.
4. Малышев Е. В., Смелов В. В. Алгоритм распознавания плагиатов кодов программ. *Труды БГТУ.* 2018. № 1 (206). С. 135–138.
5. Никитин В. Д., Иванов А. П. Разработка методов оценки сходства алгоритмов на основе графовых моделей. *Magyar Tudományos Journal*. 2018. № 23. С. 36–41.
6. Сарвар С., Кайум З. Уль, Сафьян М., Икбал М., Махмуд Я. Выявление характерных особенностей программ для борьбы с компьютерным пиратством на основе интеллектуального анализа графов. *Труды Института системного программирования РАН*. 2019. № 31 (2). С. 171–186. DOI: https://doi.org/10.15514/ispras-2019-31(2)-12
7. Шинкаренко В. И., Куропятник Е. С. Проблемы выявления плагиата и анализ инструментального программного обеспечения для их решения. *Наука та прогрес транспорту*. 2017. № 1 (67). С. 131–142. DOI: https://doi.org/10.15802/stp2017/94034
8. Khaled F., H., Al-Tamimi M. S. Plagiarism Detection Methods and Tools : An Overview. *Iraqi Journal of Science*. 2021. № 62 (8). P. 2771–2783. DOI: https://doi.org/10.24996/ijs.2021.62.8.30
9. Kulkarni S., Govilkar S., Amin D. Analysis of Plagiarism Detection Tools and Methods. *SSRN Electronic Journal*. 2021. № 1. P. 1–7. DOI: https://doi.org/10.2139/ssrn.3869091
10. Kuropiatnyk O., Shynkarenko V. Text Borrowings Detection System for Natural Language Structured Digital Documents. *COLINS 2020. Computational Linguistics and Intelligent Systems. Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Systems (COLINS 2020)* (Lviv, 23–24 April 2020). Lviv, 2020. Vol. 2604. P. 294–305.
11. Pandit A. A., Toksha G. Review of plagiarism detection technique in source code. *International Conference on Intelligent Computing and Smart Communication 2019*. 2020. P. 393–405. DOI: https://doi.org/10.1007/978-981-15-0633-8_38
12. *Programforyou : редактор блок-схем*. URL: https://programforyou.ru/block-diagram-redactor

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

13. Shynkarenko V. I., Ilman V. M. Constructive-Synthesizing Structures and Their Grammatical Interpretations. i. Generalized Formal Constructive-Synthesizing Structure. *Cybernetics and Systems Analysis*. 2014. Vol. 50. Iss. 5. P. 665–672. DOI: https://doi.org/10.1007/s10559-014-9655-z
14. Shynkarenko V. I., Ilman V. M. Constructive-Synthesizing Structures and Their Grammatical Interpretations. II. Refining Transformations*. *Cybernetics and Systems Analysis*. 2014. Vol. 50. Iss. 6. P. 829–841. DOI: https://doi.org/10.1007/s10559-014-9674-9

O. S. KUROPIATNYK[*1], B. M. YAKOVENKO[*2]

[1*]Dep. «Computer Information Technology», Dnipro National University of Railway Transport named after Academician V. Lazaryan, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail olena.kuropiatnyk@gmail.com, ORCID 0000-0003-2286-884X
[2*]Dep. «Computer Information Technology», Dnipro National University of Railway Transport named after Academician V. Lazaryan, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail bohdanyakovenko98@gmail.com, ORCID 0000-0001-6174-0027

Identification of the Program Text and Algorithm Correspondence Based on the Control Graph Constructive-Synthesizing Model

Purpose.The main article purpose is to develop and implement the method for identifying the correspondence between the text and the program algorithm represented in the form of a flowchart. As part of the method work conversion of the input data in the graph representation is performed by means of constructive-synthesizing modelling. **Methodology**. To compare the program text and flowchart, we constructed a mathematical model for converting the program code into a graphical representation on the basis of control structures. To build the model, the apparatus of constructive-synthesizing modeling and its methods were used: specialization, concretization, interpretation and implementation. The graph representation of the text is created taking into account the control operators; the flowcharts are created using a json file containing the description of the diagram elements and their links. To compare the graphs we use the breadth-first search algorithm with the number of identical vertices being counted. To obtain the software implementation of the developed method and models we used the technology of object-oriented programming and CASE-technologies, which are based on the unified modeling language UML. **Findings** A method is proposed to present the text and the flowchart of the program in a uniform format of the directed graph (control graph) and to evaluate their correspondence by the number of identical vertices. For its formalization and automated usage, we developed constructive-synthesizing models of input data transformers. The program application was developed based on the models and the method. **Originality**. The methods of constructive-synthesizing modeling in the tasks of processing texts written in artificial languages were further developed. We developed the system of constructors, which transforms text program in C++ into a control graph. **Practical value**. The results are significant for solving such tasks as assembling program texts for borrowings detection, determining the correspondence of the program algorithms and their software implementations to improve coding skills. The graph representation produced by the developed system of constructors can be used for investigation of influence of optimization and code refactoring on the program complexity using McCabe's metrics.

Keywords: constructive-synthesizing modelling; constructor; graph representation of the text; program control graph; algorithm; algorithm correspondence

REFERENCES

1. Ivutin, A. N., & Troshina, A. G. (2019). Petri-net based method of parallel programs formal verification. *Vestnik of Ryazan State Radio Engineering University, 70*, 15-26. DOI: https://doi.org/10.21667/1995-4565-2019-70-15-26 (in Russian)
2. Kondratyev, D., Maryasov, I., & Nepomniaschy, V. (2018). The Automation of C Program Verification by Symbolic Method of Loop Invariants Elimination. *Modeling and Analysis of Information Systems, 25*(5), 491-505. DOI: https://doi.org/10.18255/1818-1015-2018-5-491-505 (in Russian)
3. Cormen, Th., Leiserson, Ch., Rivest, R., & Stein, C. (2005). *Introduction to Algorithms*. Moscow: Vilyams. (in Russian)

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп'ятник, Б. М. Яковенко, 2021

4. Malyshev, Ye. V., & Smelov, V. V. (2018). Plagiarism detecting algorithms for software code. *Proceedings of BSTU, 1*(206), 135-138. (in Russian)
5. Nikitin, V. D., & Ivanov, A. P. (2018). Development of Methods for Assessing the Similarity of Algorithms Based on Graph Models. *Magyar Tudományos Journal, 23*, 36-41. (in Russian)
6. Sarwar, S., Qayyum, Z. Ul., Safyan, M., Iqbal, M., & Mahmood, Y. (2019). Graphs Resemblance based Software Birthmarks through Data Mining for Piracy Control. *Proceedings of the Institute for System Programming of the RAS, 31*(2), 171-186. DOI: https://doi.org/10.15514/ispras-2019-31(2)-12 (in Russian)
7. Shynkarenko, V. I., & Kuropiatnyk, O. S. (2017). Plagiarism detection problems and analysis software tools for its solve. *Science and Transport Progress, 1*(67), 131-142. DOI: https://doi.org/10.15802/stp2017/94034 (in Russian)
8. Khaled, F., & H. Al-Tamimi, M. S. (2021). Plagiarism Detection Methods and Tools: An Overview. *Iraqi Journal of Science, 62*(8), 2771-2783. DOI: https://doi.org/10.24996/ijs.2021.62.8.30 (in English)
9. Kulkarni, S., Govilkar, S., & Amin, D. (2021). Analysis of Plagiarism Detection Tools and Methods. *SSRN Electronic Journal*, *1*, 1-7. DOI: https://doi.org/10.2139/ssrn.3869091 (in English)
10. Kuropiatnyk, O., & Shynkarenko, V. (2020). Text Borrowings Detection System for Natural Language Structured Digital Documents. In *COLINS 2020. Computational Linguistics and Intelligent Systems. Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Systems (COLINS 2020)* (Vol. 2604, pp. 294-305). Lviv, Ukraine. (in Ukrainian)
11. Pandit, A. A., & Toksha, G. (2019). Review of Plagiarism Detection Technique in Source Code. In *Algorithms for Intelligent Systems.* (pp. 393-405). DOI: https://doi.org/10.1007/978-981-15-0633-8_38
12. *Programforyou: redaktor blok-skhem.* Retrieved from https://programforyou.ru/block-diagram-redactor (in Russian)
13. Shynkarenko, V. I., & Ilman, V. M. (2014). Constructive-Synthesizing Structures and Their Grammatical Interpretations. i. Generalized Formal Constructive-Synthesizing Structure. *Cybernetics and Systems Analysis, 50*(5), 655-662. DOI: https://doi.org/10.1007/s10559-014-9655-z (in English)
14. Shynkarenko, V. I., & Ilman, V. M. (2014). Constructive-Synthesizing Structures and Their Grammatical Interpretations. II. Refining Transformations*. *Cybernetics and Systems Analysis, 50*(6), 829-841. DOI: https://doi.org/10.1007/s10559-014-9674-9 (in English)

Надійшла до редколегії: 29.03.2021
Прийнята до друку: 30.07.2021

Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/245666
© О. С. Куроп’ятник, Б. М. Яковенко, 2021