and wear resistance of contact materials, researchers are forced to use the results obtained by switching tests of contact materials. However, as a result of the research, it was found that in order to reduce the error and obtain a more adequate model, it is advisable to use a quadratic function, the use of which reduces the error by about half.

As a result of the study of contact pairs from different contact materials, the following was revealed: the exponential function cannot be used as a model of low-current electrical contact pairs; the linear and power functions give quite acceptable results and can be recommended for use at the initial stages of modeling the process of low-current electrical contact pairs; the quadratic function is a more adequate mathematical model of.

Key words: *contact pairs, mathematical model, dynamics of electrical processes, regression analysis, electrical erosion.*

Отримано: 17.09.2020

УДК 004.832
DOI: 10.32626/2308-5916.2020-21.125-139

С. І. Шаповалова, канд. техн. наук
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ

ФОРМАЛІЗАЦІЯ ПРЕДСТАВЛЕННЯ ПРОДУКЦІЙНИХ ПРАВИЛ В ERLANG

У статті запропоновано формалізацію представлення в Erlang продукційної моделі представлення знань та відповідне представлення продукційних правил, умовна частина яких відповідає логіці першого порядку. Метою роботи є створення функції Erlang, яка не тільки представляє в базі знань правило висновування, а також виконує його активізацію при виклику.

Наведено відомості про існуючі реалізації логічного висновування в Erlang за обома підходами до формування міркувань: логічним — Erlog та продукційним: ERESYE, SERESYE та RUNES II. Обгрунтовано доцільність розроблення для Erlang власного механізму міркувань, що базується саме на властивостях цієї мови програмування.

Визначено базові принципи концепції логічного висновування за продукційною моделлю, орієнтовані на ефективне використання вбудованого механізму співставлення Erlang, для прискорення логічного висновування. В запропонованій формалізації кожна одиниця представлення має два визначення за синтаксисами логіки та Erlang. Формалізацію відповідно до рівню об’єктів представлення розподілено на три частини: визначення базових елементів логіки, представлення умов логічного висновування (зразків і фактів), представлення безпо-

середньо компонентів продукційної моделі (правила, робочої пам'яті, конфліктного набору).

На основі запропонованої формалізації продукційної моделі для Erlang розроблено функцію активації правила бази знань для ефективного логічного висновування. При виклику — у випадку успішної активації — ця функція повертає кортеж зі списку екземплярів поточного правила та показника його пріоритетності. Кожен елемент списку відповідає комбінації фактів робочої пам'яті, які були успішно узгоджені зі зразками умовної частини поточного правила. У випадку, коли жодного варіанту узгодження не існує, функція повертає кортеж з пустим списком.

Наведено приклад створення та застосування функції активації для поточного правила.

Ключові слова: *Erlang, Erlog, ERESYE, SERESYE, RUNES II, логічні міркування, експертні системи, механізм виведення.*

Сучасне програмне забезпечення неможливе без реалізації задач штучного інтелекту, зокрема послідовних міркувань. Це необхідно в багатьох прикладних галузях, насамперед, в системах діагностування та моніторингу, бортових системах керування, сервісах забезпечення інформації та послуг. Для вирішення задач логічного висновування використовується спеціальний програмний інструментарій, який містить механізм виведення заключень та надає можливість представляти правила висновування на декларативній мові. Найбільш розповсюдженими реалізаціями такого інструментарію є обгортки експертних систем та мови логічного програмування сімейства Prolog. Однак застосування такого інструментарію найбільш ефективно для створення автономних експертних систем. У багатьох випадках, наприклад, коли міркування необхідні для вирішення однієї з багатьох задач, що паралельно вирішуються, і при цьому тісно пов'язані з їх результатами, доцільно використовувати механізм логічного висновування, «вбудований» в базову мову програмного комплексу.

Erlang — одна з сучасних мов програмування, яка була створена для забезпечення телекомунікаційних систем, але в подальшому набула статусу мови загального призначення. Початкова специфіка мови Erlang зумовила властиві їй концептуальну відмовостійкість, здатність до паралельної обробки запитів в режимі реального часу, миттєвих операцій зі створення/знищення процесів та обміну повідомлень між ними. Завдяки цьому, застосування Erlang затребувано для реалізації паралельних розподілених систем, що здатні обслуговувати мільйони підключень.

Тенденція щодо інтелектуалізації всього сучасного програмного забезпечення зумовлює актуальність розроблення для Erlang власного механізму міркувань. Задача створення моделі представлення правил логічного висновування, що базується на декларативній та функціональній концепції Erlang, є актуальною і має практичну значущість.

Аналіз останніх досліджень. Для реалізації логічного висновування в Erlang використовувались обидва підходи до формування міркувань. За логічною моделлю було створено Erlog — інтерпретатор мови Prolog в Erlang; за продукційною — обгортки експертних систем: ERESYE (ERlang Expert SYstem Engine), SERESYE (Swarm oriented ERlang Expert SYstem Engine) та RUNES II. В таблицю 1 зведено відомості про ці реалізації. Останній стовпчик містить посилання як на публікації, так і на ресурси представлення програмного забезпечення. Відповідно вказується рік публікації або останнього оновлення програмного забезпечення.

Таблиця 1

Засоби реалізації логічного висновування в Erlang

№	Назва	Реалізація	Автори	Організація, в якій створено проект	Посилання
1	Erlog	Інтерпретатор мови Prolog, що інтегру-ється у вузол Erlang.	R. Virding	Erlang Solutions Ltd, Sweden	[1], останнє оновлення 2019
2	ERESYE	Бібліотека Erlang для розробки експертних сис-тем та механізмів обробки правил	C. Santoro, A. Di Stefano, F. Gangemi	University of Catania, Italy; Erlang Training and Consulting, UK	[2], 2005; [3], останнє оновлення 2015
3	SERESYE	Бібліотека Erlang для розробки експертних систем та механізмів обробки правил	E. Mirrett, P. Nimreez, Yu. Rashkovskii	Afiniate, Inc., USA	[4], останнє оновлення 2017
4	RUNES II	Система висновування на правилах для хмарних обчислень	R. Zhou, G. Wang, J. Li Jinghan Wang, Rui Zhou, Jing Li, and Guowei Wang	University of Science and Technology of China	[5], 2014

Всі зазначені засоби пропонують логічне висновування за класичними моделями, пристосовуючи для його реалізації вбудовані механізми Erlang. Так, всі обгортки розробки експертних систем [2-5], реалізують Rete алгоритм співставлення зі зразком. Алгоритми Rete [6] та TREAT [7] є базовими методами співставлення зразків антецедентів правил з фактами робочої пам’яті на етапі активації.

Саме цей етап є найбільш затратним за часом виконання. Не існує формального доведення оптимальності одного з них для поточної задачі [8]. За обома методами співставлення проводиться прекомпіляція бази знань, в результаті якої будується спеціальна графова структура представлення умов висновування. Співставлення на поточному кроці здійснюється як пошук на графі.

Однак Erlang має власний ефективний механізм співставлення, який фактично здійснює обмежену уніфікацію. Враховуючи специфіку Erlang щодо організації паралельних процесів та можливостей функціонального програмування, доцільно розробити орієнтовану на вбудований механізм співставлення модель представлення правил для пришвидшення їх активізації.

Мета і задачі. Метою роботи є створення функції, яка представляє в базі знань правило висновування і виконує його активізацію при виклику. Для виконання даної мети необхідно було вирішити такі завдання:

1. Визначити концептуальні вимоги до логічного висновування в Erlang.
2. Розробити формалізацію продукційної моделі для Erlang.
3. Створити функцію представлення правил в Erlang.
4. Реалізувати та навести приклад виконання запропонованої функції.

Концепція продукційної моделі орієнтована на ефективне використання вбудованого механізму співставлення Erlang та його властивостей функціональної мови програмування.

Концепція полягає в наступному:

1. Доведення істинності умови відбувається співставленням зі зразком: якщо зразок продукційного правила уніфікується з фактом, який знаходиться в робочій пам'яті WM (Working Memory), цей зразок вважається істинним.
2. Для співставлення використовується виключно вбудований механізм Erlang: умови логічного висновування (зразки, факти) обробляються безпосередньо і не потребують прекомпіляції, тобто не використовуються спеціальні алгоритми співставлення, такі як RETE та Treat, які потребують побудови додаткових графових структур зв'язків умов та їх аргументів.
3. Умовна частина кожного правила бази знань KB (Knowledge Base) представляється кон'юнктивною зв'язкою зразків: це відповідає приведенню антецеденту до диз'юнктивної нормальної форми і дозволяє представити кожен кон'юнкт окремою фразою Erlang.
4. Умовна частина — антецедент (antecedent) — та заключення — консеквент (consequent) — правила KB представляються в одній функції, виконання якої призводить до одного з двох результатів:

1) доведення істинності умов з одночасною уніфікацією їх змінних з відповідними аргументами консеквента,
2) доведення хибності умов.

5. Передбачено можливість наявності в WM декількох фактів, які можна співставити з одним й тим самим зразком на поточному кроці висновування, що призводить до занесення в конфліктний набор декількох екземплярів (instantiation) одного й того самого правила КВ.
6. Модель містить базові критерії визначення пріоритету правила при розв'язанні конфлікту.

Формалізацію продукційної моделі в Erlang відповідно до рівню об'єктів представлення розподілено на три частини: визначення базових елементів логіки, представлення умов логічного висовування: зразків і фактів, представлення компонентів продукційної моделі: правила, робочої пам'яті, конфліктного набору. Формалізацію скомпоновано таким чином, щоб визначення кожного поточного об'єкту не містило раніш невизначених аргументів, тобто посилалось на попередні визначення та додавало нові властивості.

Erlang концептуально пов'язаний з мовою логічного програмування Prolog насамперед механізмом співставлення, синтаксисом та використанням змінних. Це дозволяє провести пряму аналогію між логічним поданням умов і правил висновування та їх представленням в Erlang. Тому в запропонованій формалізації кожна одиниця представлення має два визначення — в логіці (ліворуч) і в Erlang (праворуч) — та одне посилання формули. У визначеннях дотримано відповідні синтаксиси. Зокрема в Erlang позначення змінних починаються з великої літери, а констант — з маленької; квадратні дужки визначають список. Зауважимо, що фігурні дужки в логіці предикатів позначають множину, а в Erlang — кортеж. У такому сенсі ці позначення використовуються в подальшій формалізації.

Введемо **базові елементи логіки**, необхідні для формалізації моделі.

Константа може бути атомом або колекцією атомів. Будь-який інший вираз, аргументами якого є тільки константи, константою не являється. Це обмеження випливає з умови, що константа не має бути результатом виразу, що виконується в функціональному програмуванні:

```
                    constant ≡                                    (1)
atomic constant |   |     atom | number |
 atomic constants   |list of atomic constants |
    collection      | tuple of atomic constants
```

Обмеження: `constant ≠ expression from constants`

Оскільки визначення атомарної константи, атому, числа, списку та кортежу повністю відповідають своїм визначенням в логіці предикатів та Erlang, в даній моделі вони не наводяться.

Змінна може бути зв'язаною або незв'язаною:

$$\text{Variable} \equiv \text{bound} \mid \text{unbound} \quad \Big| \quad \text{constant} \mid V, \tag{2}$$

де V — позначення змінної.

Позначення змінної призначається за синтаксисом Erlang.

Терм може бути константою або змінною:

$$\text{Term} \equiv \text{constant} \mid \text{Variable}, \tag{3}$$

де constant — константа (1); Variable — змінна (2).

Атомарна формула містить ім'я предикату та перелік аргументів — термів. Завдяки тій самій умові, що призвела до обмеження визначення константи, терм не може бути складеним.

$$\text{Atomic Formula} \equiv \text{predicate}(X_1, X_2, \ldots, X_n) \mid \{\text{predicate}, [X_1, X_2, \ldots, X_n]\} \tag{4}$$

де predicate — предикативний символ; X_i — терм (3).

Обмеження: term ≠ compound term

В Erlang атомарна формула представлена кортежем тому, що традиційний запис предикату в Erlang є функцією, що виконується, і не може бути застосований для представлення даних.

Літерал є позитивною або негативною атомарною формулою:

$$\text{Literal} \equiv \text{Atomic Formula} \mid \neg \text{Atomic Formula}) = \ <\neg > \text{predicate}(X_1, X_2, \ldots, X_n) \quad \Big| \quad \{\text{predicate}, [X_1, X_2, \ldots, X_n], \text{Sign}\} \tag{5}$$

де Atomic Formula — атомарна формула (4); predicate — предикативний символ; X_i — терм (3);

<¬ > — знак логічного заперечення, який використовується тільки в негативних формулах;

Sign — показник, що визначає представлена атомарна фор-мула або її заперечення:

$$\text{Sign} \equiv \text{pos} \mid \text{neg} \tag{6}$$

В логіці предикатів підстановкою Θ є кінцева множина:

$$\Theta = \{t_1/X_1, t_2/X_2, \ldots, t_n/X_n\}, \tag{7}$$

де X_i — змінна, яка позначає i-й терм, при чому всі $X_1, X_2, \ldots, X_n$ — різні; t_i — значення i-го терму (константа).

Застосування підстановки Θ до деякої формули означає заміну всіх змінних X_i на t_i. Позначимо через LiteralΘ частинний випадок літерала, отриманий після підстановки Θ.

Приклад літерала після підстановки Θ:

$$\text{Literal}\Theta = <\neg> \text{predicate}(t_1, t_2, \ldots, t_n) \quad \Big| \quad \{\text{predicate}, [t_1, t_2, \ldots, t_n], \text{Sign}\}, \tag{8}$$

де predicate — предикативний символ; t_i — i-тий терм літерала після підстановки (t_i = constant); <⌐> | Sign — наявність| відсутність заперечення (6).

Результат застосування множини підстановок $\{\Theta_1,\ldots\Theta_k\}$ до літерала позначимо $Literal\{\Theta_i\}$:

$$Literal\{\Theta_i\} = \{Literal\Theta_1,\ Literal\Theta_2,\ \ldots,\ Literal\Theta_k\} \mid [Literal\Theta_1,\ Literal\Theta_2,\ \ldots,\ Literal\Theta_k] \quad (9)$$

де $Literal\Theta_j$ — приклад літерала після j-тої підстановки Θ_j (8), j=1..k.

Останній результат стосується однієї з проблем реалізації логічного висновування за продукційною моделлю — необхідністю врахування декількох (більш ніж одного) прикладів підстановки для одного й того ж зразку. В разі існування декількох комбінацій фактів WM, які призводять до істинності умовної частини поточного продукційного правила, всі екземпляри цього правила заносяться до конфліктного набору.

Представлення умов логічного висновування. В продукційній моделі доведення істинності умов здійснюється на основі співставлення зі зразком. Кожна умова в правилі є зразком *P (P*attern), який зіставляється з заданими або доведеними фактами робочої пам'яті. У випадку успішної уніфікації відповідних зразка та факту поточна умова вважається істинною. Структура зразків відповідає фактам WM. На відміну від фактів, аргументи зразків можуть мати незв'язані змінні.

Зразки представляються літералами з додатковим аргументом — позначкою часу доведення факту TimeStamp. В продукційному правилі цей аргумент є незв'язаною змінною. У випадку успішного співставлення на поточному кроці даного зразка з фактом робочої пам'яті, TimeStamp зв'язується відповідним значенням. Цей параметр є визначальним в стратегіях розв'язання конфлікту. Зразок визначається таким чином:

$$P \equiv Literal|_{TimeStamp} = {<\!\!\lceil\!\!>}predicate(X_1, X_2, \ldots, X_n)|_{TimeStamp} \mid \{predicate,\ [X_1,\ X_2, \ldots, X_n],\ Sign,\ TimeStamp\}, \quad (10)$$

де Literal — літерал (9), predicate — предикативний символ; X_i — i-тий терм літерала (3), <⌐> | Sign — наявність| відсутність заперечення (6), TimeStamp — незв'язана змінна показнику часу доведення факту, відповідного поточному зразку.

Факт F (Fact) є — прикладом літерала після підстановки Θ, тобто факт містить тільки константи:

$$F \equiv \text{Literal}\Theta = \qquad (11)$$

$$\text{<⌐>predicate}(t1,\ldots,tn)\,|_{\text{timeStamp}} \;\Big|\; \{\text{predicate}, [t_1,\ldots,t_n], \text{Sign}, \text{timeStamp}\},$$

де $\text{Literal}\Theta$ — приклад літерала після підстановки Θ (8), predicate — предикативний символ; t_i — i-тий терм літерала (константа), <⌐> | Sign — наявність| відсутність заперечення (6), TimeStamp — зв'язана змінна показнику часу доведення факту, відповідного поточному зразку.

Всі аргументи факту є зв'язаними.

В робочій пам'яті всі факти з одним й тим самим предикативним символом і структурою представляються записами формату (11), кожен з яких має унікальний набір аргументів. Таким чином представляється множина підстановок (9).

Одиницею представлення WM є факт (11).

Представлення продукційних правил. За загальноприйнятою продукційною моделлю правило представляє зв'язок передумов та заключення:

$$\text{Antecedent -> Consequent}, \qquad (12)$$

де Antecedent — умовна частина; Consequent — частина дії.

Представлення антецеденту має відповідати логіці 1-го порядку.

Консеквент — перелік фактів, які доводяться або спростовуються за умови істинності антецеденту продукційного правила. Тобто кожен факт консеквенту — відповідно — додається або видаляється з WM. Випадки додавання доведених фактів найпоширеніші, тому такі факти в Erlang запропоновано представляти безпосередньо — без додаткових вказівок — за визначенням (11). Якщо факт має бути видаленим з робочої пам'яті, він представляється кортежем з тегом del.

Для виокремлення різниці між фактами робочої пам'яті і фактами консеквенту для останніх введемо позначення $\dot{F}$. Різниця полягає не тільки в наявності дії щодо зміни робочої пам'яті, але й в наявності в фактах консеквенту незв'язаних змінних, на відміну від фактів WM. Факт консеквенту у представленні правила є неосновним фактом. Безпосередньо фактом він стає після підстановки змінних, які визначені в зразках умовної частини і зв'язані на основі співставлення з фактами WM, відповідними умовам.

Тому у визначенні факту консеквенту використовується визначення зразку:

$$\dot{F} \equiv \qquad (13)$$

$$P^{+} \mid P^{-} \;\Big|\; P \mid \{\ \text{del}, P\ \}$$

де P — зразок (10),

P^{+} — факт, що додається, P^{-} — факт, що видаляється, | P — факт, що додається, { del, P } — факт, що видаляється.

Консеквент є множиною доведених/спростованих фактів:

$$Consequent \equiv \{ \dot{F}_1, \dot{F}_2, \ldots, \dot{F}_w \} \mid [\dot{F}_1, \dot{F}_2, \ldots, \dot{F}_w] \quad (14)$$

де F^*_j — j-й факт з визначенням дії щодо нього (13), $j=1..w$

Враховуючи декларативність Erlang та особливості його вбудованих механізмів, в запропонованій моделі для спрощення та прискорення процесу обробки продукційних правил, умовна частина кожного з них має бути представлена у диз'юнктивній нормальній формі DFN (disjunctive normal form). Тобто антецедент є диз'юнктивною зв'язкою кон'юнтів:

$$Antecedent = Conjunction_1 \vee Conjunction_2 \vee \ldots \vee Conjunction_d \quad (15)$$

де $Conjunction\ i$ — i-й кон'юнкт, $i=1..d$.

Кон'юнкт в моделі є кон'юнктивною зв'язкою зразків:

$$Conjunction \equiv \{P_1 \wedge P_2 \wedge \ldots \wedge P_m \} \mid [P_1, P_2, \ldots, P_m] \quad (16)$$

де Pj — j-й зразок (13) кон'юнкту, $j=1..m$.

Таким чином, одне продукційне правило представляється декількома клозами — фразами Erlang, кількість яких дорівнює кількості кон'юнктів:

$$Production \equiv \begin{array}{l|l} (Conjunction_1 \text{ -> } Consequent_1) \vee & Clause_1 \text{ ;} \\ (Conjunction_2 \text{ -> } Consequent_2) \vee & Clause_2 \text{ ;} \\ \ldots & \vdots \\ \vee\ (Conjunction_d \text{ -> } Consequent_d) & Clause_d. \end{array} \quad (17)$$

де

$Conjunction_i$ — i-й кон'юнкт (16), $i=1..d$; $Consequent_i$ — консеквент (14), який відповідає i-му кон'юнкту;

$Clause_i$ — фаза Erlang, яка відповідає *i*-й зв'язці умов та заключення, $i=1..d$.

Обмеження: $Antecedent \equiv logic\ connective\ in\ DFN$

В стратегіях розв'язання конфлікту, що базуються на сортуванні активацій за пріоритетами, окрім критеріїв новизни та специфічності, використовують значущість $Salience$. Цей параметр дозволяє користувачеві призначати правилам пріоритет.

Таким чином, кожен клоз бази знань має містити таку інформацію:

$$\{ Conjunction_n, Consequent_n, Salience_n \}, \quad (18)$$

де n — номер правила КВ, $Conjunction_n$ — кон'юнкт (16), $Consequent_n$ — консеквент (14), $Salience_n$ — показник пріоритетності n-го правила.

Кожен клоз може оброблятися автономно.

Функція представлення та активації продукційного правила. Уніфікація змінних логічних умов продукційного правила є основною проблемою реалізації продукційної моделі засобами Erlang.

Оскільки умовна частина кожного правила КВ є кон'юнктивною зв'язкою, доведення істинності відбувається за умови успішної уніфікації кожного з представлених зразків з фактами робочої пам'яті.

Уніфікація полягає в пошуку в робочій пам'яті факту з тим самим предикативним символом та здійсненні можливої підстановки множини аргументів. Кожен аргумент зразку умовної частини уніфікується з відповідним аргументом факту WM. Уніфікація успішна, якщо:

1) відповідні аргументи — однакові константи;
2) аргумент зразку є незв'язною змінною, а аргумент факту — константою.

В останньому випадку всі однойменні змінні антецеденту та консеквенту поточної фрази мають бути зв'язаними цією константою.

Проблема полягає в тому, що вбудований механізм співставлення Erlang, на відміну від Prolog, реалізує лише часткову уніфікацію. В даному випадку це означає, що неможливо безпосередньо представити в правилі зразки з незв'язаними змінними, які заздалегідь не визначаються в тому самому клозі Erlang або при виклику функції, якій він належить. Тому кортеж (18) представити безпосередньо неможливо. Для розв'язання цієї проблеми запропоновано спеціальну структуру, яка базується на використанні генераторів списку. Як спеціальний засіб функціонального програмування, генератор списку дозволяє задавати незв'язані змінні, які згодом будуть уніфіковані на основі співставлення з елементами відомого списку.

При заповненні бази знань програміст визначає всі зразки та факти поточного правила та незв'язані змінні в них. Якщо ці змінні вказують на один й той самий об'єкт, їм призначаються однакові позначення. Для їх зв'язування достатньо помістити обидві частини правила в один вираз, який задасть формат генерації відповіді при активації правила.

Окрім цього у відповідь має входити структура `Trace`, яка однозначно представить екземпляр правила для занесення в **конфліктний набор** `Agenda`. Така структура представляється кортежем, який містить номер правила в КВ та перелік всіх позначень незв'язаних змінних, які є в `Conjunction` і `Consequent`.

Таким чином, при активації правила генерується список відповідей зі структурою:

$$\{\ \mathrm{Trace}_n,\ \mathrm{Conjunction}_n,\ \mathrm{Consequent}_n\ \}, \qquad (19)$$

де n — номер правила КВ, Trace_n — «слід» правила, $\mathrm{Conjunction}_n$ — кон'юнкт (16), $\mathrm{Consequent}_n$ — консеквент (14).

В кортежі (19) не враховано показник пріоритетності правила $Salience_n$, оскільки цей параметр однаковий для всіх екземплярів правила і доцільніше його повернути в загальній структурі рішення.

При реалізації частини генерації необхідно представити послідовну уніфікацію кожного зразка Conjunction з відповідними фактами WM. Тобто має бути застосовано m генераторів — за кількістю зразків Conjunction. Однак область дії змінної обмежується тим генератором, де вона представлена. Тому при застосуванні декількох генераторів було використано спеціальні умови зв'язування VarsBinding (Variables Binding Conditions) на основі оператору «= =». Умови VarsBinding застосовуються, починаючи з 2-го генератора.

Окрім безпосереднього подання зразків в умовні частині за виразом (10), в модель закладено додаткову можливість здійснення порівнянь значень змінних. Це реалізовано додаванням умов генерації у форматі Guard — охоронного виразу, який містить зв'язані та незв'язані змінні поточного зразку P_i.

Назва і позначення охоронного виразу Guard запозичені в Erlang. Охоронний вираз — це логічний вираз, який представляє додаткове обмеження в голові функціональних рівнянь, if–, case– та receive– виразах. В запропонованій моделі зі всіх різновидів охоронних виразів Erlang використовуються лише операції порівняння. Для виконання цих операцій використовується безпосередньо механізм Erlang, тому:

1) збережено синтаксис Erlang;
2) відсутні обмеження на вирази, що порівнюються;
3) припускається використовувати логічні зв'язки охоронних виразів.

Таким чином, кожен $Guard_i$ є операцією порівняння або логічною зв'язкою таких операцій.

Вираз Guard не є обов'язковим, тому його назва в наведеній нижче функції (20) міститься в символах < >.

Аргументами цієї функції є номер правила КВ, позначка часу активації поточного правила (може вказуватись номер поточного циклу висновування) та список істинних фактів — WM. Номер правила фактично дублюється в «сліді» Trace, однак його виокремлено в аргументи функції з міркувань надання можливостей розпаралелювання обробки бази знань. Позначка часу активації поточного правила у разі доведення істинності умов уніфікується з усіма незв'язаними змінними позначок часу доведення фактів консеквенту. Таким чином забезпечується додавання в WM фактів з актуальним часом доведення.

Функція, яка представляє фразу КВ, має такий формат:

```
                    Clause n ≡                                    (20)
rule(N, CurTimeStamp, WM) ->
      {
            [{
Trace_n,              %% {N, UnboundVarsSet}
            Conjunction_n,      %% [ P_1, P_2, …, P_m ]
            Consequent_n        %% [ Ḟ_1, Ḟ_2, …, Ḟ_w ]
                  } ||  P_1 <- WM, <Guard>,
                  P_2 <- WM, VarsBinding_2, <Guard>,
                  ⋮
                  P_m <- WM, VarsBinding_m, <Guard>
],
Salience_n
};
```

де N — номер правила КВ, CurTimeStamp — показник часу доведення фактів Consequent$_n$, WM — робоча пам'ять, $Trace_n$ — «слід» правила, $Conjunction_n$ — кон'юнкт (16), $Consequent_n$ — консеквент (14), P_i — i-та логічна умова — зразок (10), $\dot{F}_j$ — j-й факт Consequent з визначенням дії щодо нього (13), $VarsBinding_i$ — умови зв'язування змінних i-го та всіх попередніх генераторів, Guard — умова обмеження аргументів зразків, $Salience_n$ — показник пріоритетності n-го правила.

При виклику ця функція повертає кортеж з двох аргументів: списку рішень у форматі (19) та показника пріоритетності поточного правила. Кожен елемент списку відповідає комбінації фактів робочої пам'яті, які були успішно узгоджені зі зразками умовної частини поточного правила. У випадку, коли жодного варіанту узгодження не існує, функція повертає кортеж з пустим списком.

Таким чином, кортеж (18) перетворено в вираз Erlang, який автоматично виконується при виклику.

Приклад активації правила. Для тестування реалізації запропонованої функції активації (20) створимо правило:

```
f_1(1,X) ^ f_2(X,Y,2) ^ Y>0 ^ ⌐ (f_3(Y,3,X)) -> (21)
   f_21(1,X) ^ f_22(X,Y,2) ^ ⌐ (f_23(Y,3,X)),
```

де f_i — зразки.

Це правило містить позитивні *$f_1/2$*, *$f_2/3$* та негативну *$f_3/3$* атомарні формули та додаткову умову, що обмежує незв'язану змінну *Y*.

Правилу (21) згідно з функцією (20) відповідає наступний запис КВ:

```
                rule(1, T, WM) ->                      (22)
 {
 [
{
{1,X,Y},                                      %% { Trace,
[     {f1,{1,X},pos,T1},     {f2,{X,Y,2},pos,T2},
      %% Conjunction,
{f3,{Y,3,X},neg,T3}],
[     {f21,{1,X},pos,T},     {f22,{X,Y,2},pos,T},
      %% Consequent }
{f23,{Y,3,X},neg,T}]
      }                 ||
            {f1,{1,X},pos,T1} <- WM,
            {f2,{X1,Y,2},pos,T2} <- WM, X1 == X,
Y>0,
            {f3,{Y1,3,X2},neg,T3} <- WM, Y1 == Y,
X2 == X1],
      1}.                                     %% Saliense
```

Для відображення можливості отримання декількох екземплярів цього правила для кожного зі зразків умовної частини $f_1/2$, $f_2/3$, $f_3/3$ в робочій пам'яті представимо по 3 підстановки:

```
                          WM =                         (23)
          { f1(1,x1)|2,    |    [ {f1,{1,x1},pos,2},
         f2(x1,y1,2)|1,    |     {f2,{x1,y1,2},pos,1},
        ¬f3(y1,3,x1)|0,    |     {f3,{y1,3,x1},neg,0},
f1(1,x2)|11,               |{f1,{1,x2},pos,11},
f2(x2,y1,2)|21,            |{f2,{x2,y1,2},pos,21},
¬f3(y1,3,x2)|13,           |{f3,{y1,3,x2},neg,13},
             f1(2,x2)|2,   |       {f1,{2,x2},pos,2},
          f2(x2,y2,2)|2,   |    {f2,{x2,y2,2},pos,2},
       ¬f3(y2,3,x1)|2 }    |   {f3,{y2,3,x1},neg,2} ]
```

Очевидно, що тільки 2 комбінації фактів WM забезпечують істинність кон'юнктивної зв'язки.

На рисунку 1 приведено результат виконання функції (22) на основі фактів робочої пам'яті (23). Отриманий в середовищі Erlang результат доповнено помітками, які його структурують.

Відповідь містить опис двох екземплярів правила з номером 1, а також значущість цього правила (значення 1 зазвичай встановлюється за замовчуванням). Екземпляри правила відповідають підстановкам: $\{x_1/X,\ y_2/Y\}$ та $\{x_2/X,\ y_1/Y\}$.

Отриманий результат містить повну інформацію для доповнення конфліктного набору та виконання на поточному кроці логічного висновування заключних етапів: розв'язання конфлікту та запуску обраного екземпляру правила.

```
3> r:test().
{[{{1,x1,y1},                    { [ { Trace1
  [{f1,{1,x1},pos,2},            ---
  {f2,{x1,y1,2},pos,1},           | Conjunction1
  {f3,{y1,3,x1},neg,0}],         ---
  [{f11,{1,x1},pos,0},           ---
  {f12,{x1,y1,2},pos,0},          | Consequent1 }
  {f13,{y1,3,x1},neg,0}]},       ---
  {{1,x2,y1},                     { Trace2
  [{f1,{1,x2},pos,11},           ---
  {f2,{x2,y1,2},pos,21},          | Conjunction2
  {f3,{y1,3,x2},neg,13}],        ---
  [{f11,{1,x2},pos,0},           ---
  {f12,{x2,y1,2},pos,0},          | Consequent2 } ]
  {f13,{y1,3,x2},neg,0}]}],      ---
  1}                              Saliense }
```

Рис. 1. *Приклад результату виконання функції активації правила*

Таким чином, використання запропонованої функції (20) дозволяє ефективно здійснити етап активації — самий ресурсоємний за витратами часу та пам'яті.

Висновки:

1. Визначено базові принципи концепції логічного висновування за продукційною моделлю, орієнтовані на ефективне використання вбудованого механізму співставлення Erlang та його властивості функціональної мови програмування.
2. Запропоновано формалізацію продукційної моделі для Erlang на основі його вбудованого механізму співставлення зі зразком для прискорення логічного висновування.
3. Запропоновано функцію активації правила бази знань на основі формалізації продукційної моделі для Erlang для ефективного логічного висновування.
4. Наведено приклад створення та застосування функції активації для поточного правила.

Перспективою подальших досліджень є створення механізму висновування на основі запропонованої моделі як додаткового засобу Erlang.

Список використаних джерел:

1. Virding R. Erlog — Prolog for an Erlang Application. URL: https://github.com/rvirding/erlog.
2. Di Stefano A., Gangemi F., Santoro C. Eresye: Artificial intelligence in Erlang programs. *Proc. 2005 ACM SIGPLAN Work.* Erlang — ERLANG '05, Sept. 26-28, 2005. Tallinn, 2005. P. 62-71. DOI:10.1145/1088361.1088373.
3. Santoro C. ERESYE download. URL: https://sourceforge.net/projects/eresye.

4. SERESYE — Swarm oriented Erlang Expert System Engine. URL: https://github.com/afiniate/seresye.
5. Zhou R., Wang G., Li J. RUNES II: A Distributed Rule Engine Based on Rete network in Cloud Computing. *Internatiol journal of Grid distribution computing.* 2014. Vol. 7. № 6. P. 91-110. DOI:10.14257/ijgdc.2014.7.6.08.
6. Forgy C.L. Rete: A Fast Algorithm for the Many Pattern. *Many Object Pattern Match Problem, Artificial intelligence*. 1982. Vol. 19. № 1. P. 17-37.
7. Miranker D.P. TREAT: A New and Efficient Match Algorithm for AI Production Systems. *Morgan Kaufmann Publishers.* San Francisco, 1990. P. 143.
8. Шаповалова С. І., Мажара О. О. Вибір оптимального алгоритму співставлення зі зразком при проектуванні продукційної системи. *Східно-Європейський журнал передових технологій*. 2014. Вип. 2/2 (68). С. 43-49. DOI: 10.15587/1729-4061.2015.46571.

FORMALIZATION OF THE RULES OF INFERENCE IN ERLANG

The article proposes a method of solving logical puzzles on the basis of machine learning. The method is designed for the preliminary formalization of tasks in the form of description of properties and relations between them. Because each property has a set of possible values, the solution of the puzzle by the methods of search has a combinatorial complexity. With a large number of properties and their values, the time of the solving is rapidly increasing.

In recent years, a separate area of research in machine learning has been the solution to logical tasks of this type. However, existing solutions to this area have a number of shortcomings, first and foremost, they do not always guarantee a correct solution.

The paper presents a special network of connections for learning the solution of logical puzzles, as well as their formalization for the representation of this network. The network contains computing nodes that represent the relationship between properties, and the nodes of the input layers that specify the values of these relationships.

Every task is solved by automatically creating a network of links with its further training until the solution is obtained. The geometric interpretation of the n-dimensional network of bonds and its (n – 1)-dimensional layers is given. The formalization of the presentation of the study sample and the learning algorithm are presented. The mechanism of solving logical combinatorial problems is presented.

The article presents examples of tasks that are traditional tests in systems of logical programming and production (expert) systems, as well as tasks from the resource bAbI of such classes: two supporting facts, two argument relations, positional reasoning.

The efficiency of the proposed method has been experimentally proved.

The prospects of further researches, which are connected with the creation of a lexical-syntactic analyzer for automatic representation of properties, their values and relations between them, are determined.

The proposed method is universal and does not depend on the characteristics of the current task, such as the number of properties and their values.

Key words: *Erlang, Erlog, ERESYE, SERESYE, RUNES II, logical reasoning, rule-based systems, inference engine.*

Отримано: 21.05.2020