8. Anokhin, A. N. The system approach to analysis and description of operator activity [Text] / A. N. Anokhin // Cybernetics and Systems. – 2008. – Vol. 1. – P. 82–87.
9. Lavrov, E. Organizational approach to the ergonomic examination of E-learning modules [Text]. / E. Lavrov, O. Kupenko, T. Lavryk, N. Barchenko // Informatics in Education – an International Journal. – 2013. – Vol. 12, Issue 1. – P. 107–124.
10. Репринцева, Г. А. Системно-деятельностный подход: общенаучный и психолого-педагогический уровни анализа [Электронный ресурс] / Г. А. Репринцева // Концепт. – 2014. – № 8. – Режим доступа: http://e-koncept.ru/2014/14225.htm
11. Адаменко, А. Н. Информационно-управляющие человеко-машинные системы: Исследование, проектирование, испытания [Текст]: справочник / А. Н. Адаменко, А. Т. Ашеров, И. Л. Бердников и др.; под общ. ред. А. И. Губинского, В. Г. Евграфова. – М.: Машиностроение, 1993. – 528 с.
12. Лавров, Е. А. Автоматизация оценки условий труда на рабочем месте человека-оператора [Текст] / Е. А. Лавров, Н. Б. Пасько // Вісник Одеської державної академії будівництва та архітектури. – 2009. – № 36. – С. 250–256.
13. Lavrov, E. Computer Simulation of Systems "Man-Machine": Achievements and Tasks [Text] / E. Lavrov // Materials International Scientific Conference "UNITECH '07". – Gabrovo, Bulgaria. – 2007. – Vol. 3. – P. 358–362.

В статті в єдиному форматі представлено формалізований опис продукційної системи, компіляцію Rete та Treat мережі потоку даних, а також правил переходу між вузлами Treat мережі. Запропоновано формалізацію співставлення за Treat алгоритмом, яка може бути використана для подальшої оцінки складності. Розширено модель розрахунку витрат пам'яті для Treat алгоритму

Ключові слова: співставлення зі зразком, формалізація, Rete алгоритм, Treat алгоритм, продукційна система, логічне виведення

В статье в едином формате представлено формализованное описание продукционной системы, компиляция Rete- и Treat-сети потока данных, а также правил перехода между узлами Treat сети. Предложена формализация Treat алгоритма сопоставления, которая может быть использована для дальнейшей оценки средней сложности. Расширена модель расчета затрат памяти для Treat алгоритма

Ключевые слова: сопоставление с образцом, формализация, Rete алгоритм, Treat алгоритм, продукционная система, логический вывод

УДК 004.825
DOI: 10.15587/1729-4061.2015.46571

ФОРМАЛІЗАЦІЯ БАЗОВИХ АЛГОРИТМІВ СПІВСТАВЛЕННЯ ЗІ ЗРАЗКОМ В ПРОДУКЦІЙНИХ СИСТЕМАХ

С. І. Шаповалова
Кандидат технічних наук, доцент*
E-mail: lana@ aprodos. aprodos.kpi.ua
О. О. Мажара
Аспірант*
E-mail: olyamazhara@gmail.com
*Кафедра автоматизації проектування енергетичних процесів і систем
Національний Технічний Університет України «Київський політехнічний інститут»
пр. Перемоги, 37, м. Київ, Україна, 03056

1. Вступ

Продукційна модель представлення знань – один з найбільш часто використовуваних формалізмів вирішення задач штучного інтелекту. Створення прикладних продукційних систем потребує оцінки їх ефективності в процесі розробки. Для визначення об'єктивних критеріїв оцінювання таких систем та їх коректного порівняння необхідно застосовувати єдиний формальний опис.

Існує декілька підходів до формалізації продукційних систем [1–3]. Однак найбільш універсальним з них є формалізація на основі логіки першого порядку. В той же час формальний опис продукційної системи (ПС) в термінах логіки першого порядку наразі представлено лише частково для окремих базових механізмів обробки продукцій.

Логічне виведення в продукційних системах реалізується на основі механізмів розв'язання конфлікту та співставлення зі зразком. При цьому останній має найбільший вплив на швидкодію та затрати пам'яті в прикладній продукційній системі.

Для об'єктивного та незалежного від умов реалізації оцінювання алгоритмів співставлення необхідно представити їх формальний опис в єдиному форматі. Це дозволить визначити критерії щодо використання ресурсів пам'яті та швидкодії на стадії проектування прикладних продукційних систем за заданими характеристиками записів бази знань. Тому єдина формалі-

зація базових алгоритмів співставлення зі зразком є актуальною та має практичне значення.

2. Аналіз літературних даних і постановка проблеми

Оцінювання ресурсоємності прикладних продукційних систем здійснюється при створенні прототипу та тестуванні їх альфа версій [4] або на основі якісних критеріїв продукцій, що описують задачу логічного виведення [5]. Проте на сьогодні не представлено єдиної методики розрахунку затрат пам'яті на етапі проектування. Для її розробки необхідне створення єдиної формалізації алгоритмів співставлення, які мають найбільший влив на ресурсоємність прикладної продукційної системи.

В роботах [6] показано, що найбільш поширеним програмним інструментарієм розробки продукційних систем є CLIPS, Jess, Soar, Drools. Тому в даному дослідженні для подальшої формалізації обрано Rete та Treat алгоритми співставлення як базові для зазначених середовищ розробки. Концепції цих алгоритмів були закладені Ч. Форгі (Forgy) [7] та Д. Міранкером (Miranker) [8] відповідно та надалі лише зазнавали модифікацій у відповідності до специфічних задач [9–11].

В роботі Л. Альберта (Albert) [12] наведено формалізацію продукційної моделі в термінах логіки першого порядку для розрахунку середньої складності Rete алгоритму. Це дозволило в роботі [13] на основі даної моделі формалізувати співставлення за Rete алгоритмом з правилами переходу між вузлами Alpha- та Beta-мереж потоку даних. Запропонований для Rete математичний апарат доцільно застосувати до спорідненого Treat алгоритму. Однак для цього і необхідно представити аналогічну до Rete формалізацію Treat алгоритму.

3. Мета і завдання дослідження

Метою даної роботи є формалізація Treat алгоритму співставлення зі зразком в термінах логіки першого порядку.

Для досягнення зазначеної мети було поставлено наступні задачі:

– описати схеми співставлення за Rete та Treat алгоритмоми;

– представити формалізацію компіляції мережі потоку даних Treat алгоритму;

– представити формалізацію правил переходу в Treat мережі;

– представити формулу розрахунку витрат пам'яті для Treat алгоритму на основі запропонованої формалізації.

4. Rete алгоритм

Робота інкрементних алгоритмів співставлення ґрунтується на запам'ятовуванні кортежів фактів, які узгоджуються з частиною антецеденту. В Rete алгоритмі [7] на кожному кроці виведення зберігається узгодження умов антецеденту з фактами робочої пам'яті, а також результат зв'язування змінних.

Співставлення з допомогою інкрементних алгоритмів відбувається у два етапи. На першому етапі – компіляції – на основі умовних частин правил, що належать до БЗ, будується мережа потоку даних. Компіляція виконується один раз при запуску системи та повторюється лише у випадку внесення змін до БЗ.

На другому етапі – виконання – відбувається обробка мережі потоку даних на основі змін робочої пам'яті. Результатом цього етапу є конфліктна множина (CS) з продукцій БЗ та кортеж фактів, які призвели до активації. Пара з умовного елементу та факту, який з ним узгоджується, називається маркером (token). Алгоритми інкрементного співставлення відрізняються підходами до формування мережі потоку даних та її обробки.

Мережа потоку даних Rete алгоритмів складається з двох частин, які відповідають етапам обробки антецеденту продукції: Alpha- та Beta-мережі. Alpha-мережа, яку також називають деревом розбору, будується на основі умовних елементів, які знаходяться в межах одного шаблону. Вузли Alpha-мережі можуть містити тести декількох видів: перевірку типу, перевірку умов, перевірку зв'язків (коли одна й та сама змінна зустрічається в умовному елементі більше одного разу). Такі тести називають внутрішніми.

Beta-мережа будується на основі бінарного поєднання листових вузлів Alpha-мережі. В вузлах Beta мережі відбувається узгодження змінних для різних шаблонів одного правила. Такі тести називають зовнішніми. Вузли Beta мережі мають два входи та зберігають два типи пам'яті відповідно – ліву та праву. Виділяють два типи таких вузлів – однотипні (Any) та негативні (Not). В однотипних вузлах відбувається узгодження між виключно позитивними чи негативними умовними елементами. Негативні вузли містять тести між позитивним та негативним умовним елементом.

Термінальною є Beta-вершина, яка отримує на вході екземпляр антецеденту (instance) – повний кортеж узгоджених фактів. Вихід термінальної вершини – активоване правило, додається до конфліктного набору.

На рис. 1 запропоновано схематичне зображення схеми потоку даних (dataflow net) для Rete алгоритму.

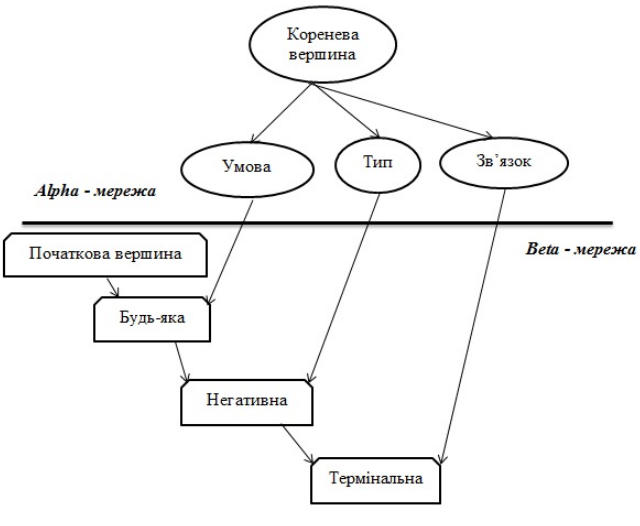

Рис. 1. Схема потоку даних Rete алгоритму

На етапі виконання Rete алгоритму зміни робочої пам'яті надходять до вузлів мережі. Якщо факт узгод-

жується з умовним елементом вузла Alpha – мережі, формується маркер і передається нащадкам вузла. В вузлах Beta – мережі відповідні маркери перевіряються на узгодження змінних в двох різних шаблонах одного антецеденту. Результати перевірки передаються далі по мережі. Досягнення листового (термінального вузла) означає активацію правила. Таким чином, в термінальних вузлах зберігається поточна конфліктна множина.

Обробка мережі після видалення фактів проводиться аналогічно додаванню. Інформація поширюється по мережі від вершини до листового вузла і призводить до деактивації відповідного правила.

5. Treat алгоритм

Treat алгоритм [8] створено на базі Rete. Treat алгоритм передбачає збереження узгодження фактів, проте зв'язування змінних обчислюється повторно на кожному кроці співставлення. Для виконання внутрішніх тестів в антецеденті будується Alpha-мережа, в вузлах (Anode) якої містяться тести для окремих шаблонів та факти, які призвели до активації поточної вершини. Підмножина таких фактів називається лівою (альфа) пам'яттю. Компіляцію внутрішніх тестів в мережу потоку даних можна представити єдиним чином для обох алгоритмів.

На відміну від Rete, Treat алгоритм не будує проміжних вершин для зв'язування змінних. Відповідні обчислення повторюються на кожному кроці співставлення. Якщо правило було активоване на попередньому кроці, то в термінальних вершинах Treat зберігається підмножина фактів, яка призвела до його активації. Таким чином, алгоритм передбачає збереження конфліктної множини.

На рис. 2 запропоновано схематичне зображення схеми потоку даних (dataflow net) Treat алгоритму.

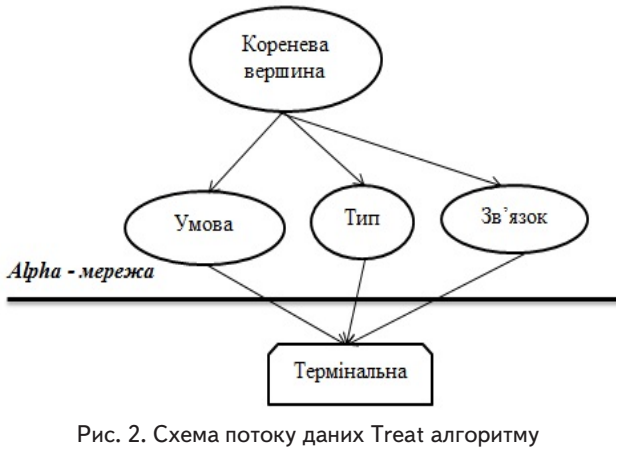

Рис. 2. Схема потоку даних Treat алгоритму

Виконання Alpha мережі Treat аналогічне до Rete алгоритму. Наступним кроком є зв'язування змінних, яке за необхідності відбувається в термінальних вершинах. На вхід до термінальної вершини надходять результати виконання альфа мережі відповідної продукції. Таким чином, на відміну від бінарних Beta вершин Rete мережі, термінальні вершини отримують на вхід кількість маркерів, яка відповідає кількості умовних елементів антецедента. Узгодження змінних відбувається попарно для кожного шаблону в довільному порядку. У найпростішому випадку реалізації обирають порядок слідування умовних елементів в записі антецеденту. Частіше використовують підходи для оптимізації порядку узгодження. У роботі [8] доведено ефективність евристики впорядкування seed ordering, за якої першим проводиться узгодження змінних для шаблону, який був активований новим фактом.

На відміну від Rete, в Treat алгоритмі додавання і видалення фактів з робочої пам'яті не є симетричним процесом. Для видалення факту в Treat мережі перевіряється конфліктна множина, збережена на попередньому кроці. Якщо один з маркерів конфліктної множини містив даний факт, то відповідні йому правила деактивуються. Інформація про видалення маркеру передається предкам відповідної альфа мережі. Таким чином вдається уникнути надлишкових узгоджень. Проте даний підхід виконується лише у випадку, коли видаляється позитивний факт.

Для обробки негативного факту він приймається за позитивний і передається на вхід до альфа мережі. Якщо це призводить до активації правила, яка міститься в конфліктній множині, правило деактивується. Продукції, які активувалися видаленням негативного факту, потребують додаткової перевірки. Alpha пам'ять негативних умовних елементів перевіряється на наявність факту, подібного до видаленого. Якщо такий факт відсутній, правило активується.

6. Формалізація продукційної системи

Продукційна система складається з трьох основних компонент: бази знань (БЗ), робочої пам'яті, механізму логічного виведення. База знань (R) – це набір продукційних правил, які є одиницями представлення інформації в продукційних системах. Робоча пам'ять (WM) – динамічна область пам'яті, в якій інформація про розв'язання поточної задачі представлена у вигляді фактів, доведених на поточному етапі виведення.

Для проведення подальших досліджень було прийнято формалізацію продукційної системи, представлену в роботах [12, 13].

Відповідно до роботи [13], в продукційних мовах програмування для забезпечення ефективності виведення використовується поняття обмеженої продукційної системи. В даній роботі використовується визначення так званої S^4 обмеженої продукційної системи:

$$PS = (F, P, X, L, WM_0, R, S),$$

де F – множина функціональних символів, P – множина предикативних символів, X – множина змінних, L – множина міток, WM0 – початковий стан робочої пам'яті, обмежений до термів висотою 1, R – множина продукційних правил, де правила мають шаблони висотою 1, S – стратегія розв'язання конфліктів.

Висота терму визначається наступним чином:

$$h(t) = \begin{cases} 0, & \text{якщо t змінна або проста константа} \\ 1 + \max_{i \le n}(h(s_i)), & \text{якщо } t = f(s_1, ..., s_n) \text{ для } n \ge 1. \end{cases}$$

Продукція – правило бази знань, яке визначається двома базовими частинами. Умовна – антецедент, який представляє собою логічну формулу, що містить зразки фактів в тому вигляді, в якому вони представляються в робочій пам'яті. В консеквенті – постумові – представлено набір дій, спрямованих на модифікацію робочої пам'яті. В загальному випадку це додавання або видалення фактів.

Продукційне правило (продукція) визначається наступним чином:

[l] if p,c remove r add a,

де l – ім'я з множини міток, p – множина позитивних та негативних шаблонів правила, c – пропозиція (proposition), множина вільних змінних якої є підмножиною змінних шаблону p, r – множина термів, екземпляри яких видаляться з робочої пам'яті після запуску правила, a – множина термів, екземпляри яких додадуться до робочої пам'яті після запуску правила.

7. Формалізація компіляції Treat мережі

Для формалізації компіляції бази знань у мережу потоку даних Treat алгоритму введемо позначення відповідно до роботи [13].

Позиція w підтерма s_i в термі t визначається наступним чином:

$$w(t)|t=0,$$
$$w(s_i)|t(s_1,...,s_n)=i,$$
$$w_1=w(s)|t \wedge w_2=w(t)|r \Rightarrow w(s)|r=w_1.w_2.$$

Тоді компіляція comp продукції $[l]p,c \Rightarrow r,a$ для Alpha-мережі Treat алгоритму аналогічна до відповідного інкрементного визначення компіляції Rete алгоритму [8]:

$$comp(t):=comp(t,0) \text{ для } t\in T(X,P), \quad (1)$$

$$comp(t(s_1,...,s_n)):=input.w=t, \quad (2)$$

$$\bigwedge_{i\in I} comp(s_i,w.i),$$

$$\forall i<j\in I \quad \bigwedge_{\exists w_1,w_2 \,|\, s_i|w_1=s_j|w_2} input.w.w_1=input.w.w_2,$$

$$comp(\neg t(s_1,...,s_n)):=\neg(\exists input\in WM)|comp(t(s_1,...,s_n)). \quad (3)$$

Для подальшої формалізації правил переходу в Alpha-мережі Treat алгоритму позначимо кон'юнкцію всіх виразів вигляду (1) – (3) як q_i^k для шаблону p_i правила l_k бази знань PM, де $k\in K$ – ідентифікатор правила, J – кількість правил в БЗ, $i\in I$ – номер шаблону в правилі, I – максимальна кількість шаблонів в правилі.

Мережа потоку даних для узгодження змінних в Treat алгоритмі формується динамічно за необхідності. Формально представлення зовнішніх тестів аналогічне до Beta мережі Rete алгоритму. Відмінністю Treat алгоритму є те, що перевірка узгодження змінних відбувається лише в рамках антецеденту окремої продукції, в якій успішно виконано всі внутрішні тести.

Наведемо компіляцію зовнішніх тестів Treat на базі відповідного формального представлення Beta мережі Rete алгоритму [13]:

$$comp(p,c):=\bigwedge_{i\in I} comp(p_i), \quad (4)$$

$$\forall i<j\in I(p^+)\vee I(p^-) \bigwedge_{\forall w_1w_2 \,|\, p_i|w_1=p_j|w_2\in X} input_i.w_1=input_2.w_2, \quad (5)$$

$$\forall i\in I(p^+)\wedge j\in I(p^-) \bigwedge_{\forall w_1w_2 \,|\, p_i|w_1=p_j|w_2\in X\wedge\neg(\exists f\in WM)} input_i.w_1=f.w_2, \quad (6)$$

$$\forall i<j\in I \bigwedge_{\exists w \,|\, p_i|w=x\in Var(c)} input_i.w=x=:\sigma_i\wedge\sigma_i(c), \quad (7)$$

де input факт, який узгоджується з поточним шаблоном, відповідно $input_i$ для шаблону p_i; p^+ – позитивний шаблон з множини I шаблонів висотою 1, p^- – негативний шаблон; X – множина змінних бази даних; $Var\in X$ – множина змінних поточного антецеденту; σ – підстановка для даного шаблону.

Кон'юнкцію всіх виразів вигляду (4)–(6) для продукції l_k бази знань PM позначимо як r^k.

8. Формалізація правил переходу у Treat мережі

В даній роботі розроблено формалізацію правил переходу в Treat мережі потоку даних. Для подальшого коректного порівняння алгоритмів співставлення правила переходу представлено за форматом та синтаксисом, запропонованим в [13].

В загальному вигляді правило переходу для вершин Treat мережі мають наступний вигляд:

$$?m \xrightarrow{a} ms!,$$

де ?m – вхідний маркер у вигляді $<+|-,f>$ для факту f, що додається чи видаляється з робочої пам'яті; ms! – список вихідних маркерів, a – опціональні дії на поточному кроці.

Для антецеденту правила π ліву пам'ять, lm, визначимо наступним чином [13]:

$$lm_i=\{(f_1,...,f_{j_i})\in WM|comp(\pi)(f_1,...,f_{j_i})\}.$$

Аналогічно визначимо термінальну пам'ять tm, яка використовується в процесі узгодження змінних та пам'ять конфліктної множини cm, яка зберігається на кожному кроці співставлення.

$$tm_i=\{f\in WM|q_{i+1}(f)\},$$

$$cm^k=\{(f_1,...,f_{j_i})\in WM|comp(l_k).$$

Для Anode правило переходу визначається наступним чином:

$$<+,f>? \xrightarrow{lm_j\leftarrow lm_j\cup\{f\}} \{+,f\}! \text{ if } q(f), \quad (8)$$

$$<-,f>? \xrightarrow{lm_j\leftarrow lm_j-\{f\}} \{-,f\}! \text{ if } q(f), \quad (9)$$

$$<+|-,f>? \rightarrow 0! \quad \text{if } \neg q(f). \quad (10)$$

Для Tnode правила переходу визначаються в залежності від типу шаблону, що узгоджується з вхідним фактом. Вихідними даними термінальної вершини є правило для активації/деактивації та відповідно факти, які мають додатися/видалитися з екземплярів конфліктного набору.

В загальному випадку при додаванні фактів перехід з термінальної вершини виконується у випадку, коли сформовано таку термінальну пам'ять, що $tm^k = cm^k$. Тобто:

$$<+,lm_i^k> \rightarrow \{<+,lm_{i_t}^k>\}! \,|\, r^k(l^k, lm_{i_t}^k). \quad (11)$$

Видалення факту зазвичай (однак не завжди) призводить до деактивації правила:

$$<-,f>? \xrightarrow{tm=\theta} \{<-,(l^k,f)>\}! \text{ if } (l^k \in CS) \wedge (f \in cm). \quad (12)$$

Можливі два випадки в залежності від того, позитивний чи негативний шаблон узгоджується з фактом, який видалено. Видалення факту, який узгоджувався з позитивним шаблоном правила:

$$<-,f>? \xrightarrow{tm=\theta} \{<-,(l^k,f)>\}!$$
$$\text{if } (l^k \in CS) \wedge (\sigma p_i^k = f \,|\, p_i^k \in p^+). \quad (13)$$

Видалення факту, який призводив до узгодження з негативним шаблоном правила:

$$<-,f>? \xrightarrow{tm=\theta} \{<+,l^k>\}! \text{if } \neg(\exists f` \in lm_i^k \,|\, \sigma p =$$
$$= f` \hat{} r(l,f`) \wedge \sigma p_i^k = f \,|\, p_i^k \in p-. \quad (14)$$

Термінальна пам'ять формується в процесі узгодження змінних в межах антецеденту. Надамо формальне представлення переходів в межах динамічної мережі узгодження при обробці термінальної вершини. Формалізуємо додавання факту до робочої пам'яті для різних випадків.

Додавання факту, який узгоджується з позитивним шаблоном:

$$<+,f>? \xrightarrow{tm_j \leftarrow tm_j \cup \{f\} \wedge \sigma \leftarrow \sigma \cup \sigma' | \exists p_i^k \in p^+(l^k) \sigma' p_i^k = f} \{<+,f>\}! \text{if } r_k(f), \quad (15)$$

$$<+,f>? \xrightarrow{tm_j \leftarrow tm_j \cup \{f\} \wedge \sigma \leftarrow \sigma \cup \sigma' | \exists p_i^k \in p^+(l^k) \sigma' p_i^k = f} \{<+,(l^k,tm_j)>\}! \text{if } r^k(tm_j), \quad (16)$$

де σ підстановка (визначена в стандартних термінах логіки першого порядку), p – шаблон правила, – позитивний шаблон, p^- – негативний.

Додавання факту, який узгоджується з негативним шаблоном:

$$<+,f>? \xrightarrow{tm_j=\theta} \{<-,(l^k,f)>\}! \text{if } r_k(f) \wedge$$
$$\wedge \exists p_i^k \in p^-(l^k) \,|\, \sigma' p_i^k = f. \quad (17)$$

Тоді для конфліктної множини CS виконуються наступні зміни у відповідності до вхідних даних з термінальної вершини:

$$<-(l^k,f)>? \xleftarrow{cm \leftarrow cm \cup \left(cm^k - \bigcup_{\forall j \neq k | cm^j \in cm} cm^j \right)} \{<-,l^k>\}!, \quad (18)$$

$$<+(l^k,tm^k)>? \xrightarrow{cm \leftarrow cm \cup tm^k} \{<+,(l^k,tm^k)>\}!, \quad (19)$$

$$<+,l^k>? \longrightarrow \{<+,l^k>\}!. \quad (20)$$

Таким чином, формули (8)–(10) описують процес обробки змін робочої пам'яті в Alpha-мережі Treat алгоритму, формули (11)–(17) – узгодження змінних в термінальних вершинах, формули (18)–(20) – формування конфліктної множини на основі результатів співставлення.

9. Розрахунок витрат пам'яті Treat алгоритму

Запропоноване формальне представлення Treat алгоритму дозволяє розширити модель розрахунку витрат пам'яті, запропоновану в роботі [14]. В цій роботі вважається, що затрати пам'яті на обробку Treat мережі потоку даних дорівнюють загальному розміру Alpha пам'яті:

$$M(n) = \sum_{j=0}^{n} a_j = \sum_{j=0}^{n} h(lm_j).$$

Однак, формалізація правил за запропонованими формулами (11)–(16) переходу показує, що на кожному кроці співставлення необхідна додаткова пам'ять, яка забезпечує збереження даних під час узгодження змінних в термінальних вершинах мережі.

Позначимо ймовірність узгодження поточного умовного елементу з наступним в черзі в межах антецеденту як $prob_j$. Тоді витрати пам'яті на обробку термінальної вершини можна обрахувати наступним чином:

$$T(i) = h(lm_0) \prod_{j=1}^{m} h(lm_j) prob_j,$$

де n – кількість правил в базі знань, h – висота (розмір) факту, $m \le n$ – кількість фактів у правилах, для яких узгодилася альфа мережа на поточному кроці співставлення, $prob_j$ – ймовірність узгодження умовних елементів в процесі співставлення.

Додатково для забезпечення асиметричного механізму видалення фактів, які узгоджувалися з позитивними умовними елементами, в Treat алгоритмі на кожному кроці зберігаються факти, які призвели до активації правил в конфліктній множині: $h(cm)$. Правила переходу та відповідні зміни для пам'яті конфліктної множини представлено формулами (17)–(19).

З урахуванням вищезазначених витрат пам'яті в процесі співставлення, формула розрахунку ресурсоємності Treat алгоритму набуває вигляду:

$$M(n) = \sum_{j=0}^{n} (h(lm_j) + T(i) + h(cm)), \quad (21)$$

де $h(lm_j)$ – затрати на збереження Alpha пам'яті, $T(i)$ – витрати пам'яті на обробку термінальної вершини, $h(cm)$ – витрати на збереження пам'яті конфліктної множини.

10. Висновки

1. В єдиному форматі описано процеси співставлення в продукційних системах за Rete та Treat алгоритмами.

2. На базі формального опису продукційної системи в термінах логіки першого порядку запропоновано формальне представлення компіляції та правил переходу в мережі потоку даних Treat алгоритму.

3. На основі розробленої формалізації запропоновано розрахунок витрат пам'яті для Treat алгоритму, який враховує витрати на обробку Alpha-мережі, збереження конфліктного набору та узгодження змінних в термінальних вершинах мережі потоку даних.

4. Формальне представлення Treat алгоритму в подальшому може бути використане для оцінювання його середньої складності на основі кількості необхідних узгоджень в антецеденті продукцій.

Література

1. Жежко, Л. Системы искусственного интеллекта. Представление знаний в информационных системах. Т. 1 [Текст] / Л. Жежко, А. Карпик и В. С. Хорошилов. – Новосибирск: СГГА, 2005. – 84 с.
2. Kowalski, R. Towards a Logic-based Production System Language [Electronic resource] / R. Kowalski, F. Sadri. – London, 2010. – Avaialble at: http://www.doc.ic.ac.uk/~fs/Papers/Reactive%20systems/LPS%20technical%20report.pdf
3. Булкин, В. Формальное представление знаний в продукционных системах [Текст] / В. Булкин, Н. В. Шаронова // Искусственный интеллект. – 2006. – № 1. – С. 147–157.
4. Tao, Y. Performance evaluation of the inference structure in expert system [Text] / Y. Tao, H. Zhijun, Y. Ruizhao // Artificial intelligence: Proceedings of the 10th international joint conference. – Kaufmann Publishers Inc. – San Francisco, 1987. – P. 945–950.
5. Шаповалова, С. І. Вибір оптимального алгоритму співставлення зі зразком при проектуванні продукційної системи [Текст] / С. І. Шаповалова, О. О. Мажара // Східно-Європейський журнал передових технологій. – 2014. – Т. 2, № 2 (68). – С. 43–49. doi: 10.15587/1729-4061.2014.23338
6. Джаррантано, Дж. Экспертные системы: принципы разработки и програмирование [Текст] / Дж. Джаррантано , Г. Райли; 4-е издание; пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 1152 с.
7. Forgy, C. L. On the Efficient Implementation of Production System : PhD thesis [Text] / C. L. Forgy. – Computer Science Department, Carnegie Mellon University. – Pittsburg, 1979. – 356 p.
8. Miranker, D. P TREAT: A New and Efficient Match Algorithm for AI Production Systems [Text] / D. P. Miranker. – London: Pitman/Morgan Kaufmann, 1990. – 144 p.
9. Liu, D. Rule Engine based on improvement Rete algorithm [Text] / D. Liu, T. Gu, J. Xue // The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding, 2010. – P. 346–349. doi: 10.1109/icacia.2010.5709916
10. Berstel, B. Extending the RETE algorithm for event management [Text] / B. Berstel // Proceedings Ninth International Symposium on Temporal Representation and Reasoning, 2002. – P. 49–51. doi: 10.1109/time.2002.1027472
11. Kang, J. A. Shortening matching time in OPS5 production systems [Text] / J. A. Kang // IEEE Transactions on Software Engineering. – 2004. – Vol. 30, Issue 7. – P. 448–457. doi: 10.1109/tse.2004.32
12. Albert, L. Average case complicity analyzes of the Rete multi-pattern match algorithm [Text] / L. Albert, F. Fages // Automata, Languages and Programming 5th International Colloquium Tampere. – Finland, 1988. – P. 18–37. doi: 10.1007/3-540-19488-6_104
13. Cirstea, H. Production Systems and Rete Algorithm Formalization [Electronic resource] / H. Cirstea, C. Kirchner, M. Moossen, P. E. Moreau. – Lorraine, 2004. – Available at: https://hal.inria.fr/file/index/docid/280938/filename/rete.formalisation.pdf
14. Wright, I. The execution kernel of RC++: RETE*, a faster RETE with TREAT as a special case [Text] / I. Wright, J. A. R. Marshall // International Journal of Intelligent Games and Simulation. – 2003. – Vol. 2, Issue 1. – P. 36–48.