УДК 618.3

В.Ю. Вінник, асист.
Житомирський інженерно-технологічний інститут

КОМПОЗИЦІЙНА СЕМАНТИКА МОВИ РЕФАЛ

Мова символьної обробки Рефал розглядається в методологічному середовищі експлікативного програмування. Виявлено найважливіші, сутнісно визначальні характеристики мови Рефал та базові структури пов'язаної з нею логіки програмування. Результати неформального аналізу мови уточнено до формалізації у вигляді композиційної системи.

Однією з галузей програмування, що мають надзвичайно велике практичне та теоретичне значення, є символьна обробка (СО), яка охоплює задачі синтаксичного аналізу та компіляції і цілий спектр задач штучного інтелекту: автоматизований переклад та реферування, доведення теорем у логіко-математичних численнях та ін. Задача побудови адекватної теоретичної моделі СО розпадається на ряд підзадач дослідження різних відгалужень та аспектів СО, доповнених метазадачею інтеграції відповідних моделей. Деякі результати в цьому напрямку викладено в роботах автора [1–7]. Цю статтю присвячено аналізу мови Рефал — широко відомої функціональної декларативної мови СО — з позицій експлікативного програмування (ЕП).

Мову Рефал було створено В.Ф.Турчиним у 1960-х роках [8–10], а у 1970–80-і роки вона активно використовувався для створення індустріальних програмних систем [11]. Як зазначають різні автори [11, 12, 13], незважаючи на свій вік, Рефал досі не втрачає практичного значення. Серед причин його довголіття називають чітку теоретичну базу — насамперед, теорію нормальних алгоритмів А.А.Маркова, завдяки якій Рефал характеризують як "мову прикладної конструктивної математики" [14 с. 118], ретельну розробку проекту мови, простоту та концептуальну цілісність, наявність ефективних трансляторів та велику вагу нечисельних задач у практиці програмування. Свідченнями нової хвилі популярності Рефалу є поява низки сайтів, зокрема сайту співдружності Рефал-Суперкомпіляція [15], а також розвиток проекту "Вбудований Рефал" (Embedded Refal) [16], орієнтованого на роботу з форматом документів XML у середовищі World Wide Web.

Розробка Рефалу як реального інструменту програмування від самого початку базувалася на математично точній і водночас прозорій семантиці [8, 9]. Проте, ця семантика відображає лише екстенсиональний, формально-математичний рівень мови і не враховує її інтенсиональної, прагматико-орієнтованої складової. Зокрема, повністю залишається за кадром найголовніший з точки зору ЕП аспект — логіка побудови програм у середовищі Рефалу.

Базові відомості з КП та ЕП

Основною задачею ЕП [17, 18, 19] є розробка теоретичних моделей реального програмування, в повній мірі адекватних його сутності — *експлікація*. Центральне місце в ЕП посідає принцип *програмологічності*: сутнісним ядром програмування є (прагматико-обумовлена — принцип *прагматичності*) *логіка* програмування — сукупність понять і засобів оперування поняттями, характерних для процесів побудови програмних об'єктів. Для того, щоб бути програмологічним засобом, програмобудівний засіб повинен узгоджуватися з інтуїтивною логікою, внутрішньо притаманною прикладній галузі. Програмний об'єкт, що розглядається крізь призму логіки побудови, називається *програмологічним*. Властивості програмологічних об'єктів у ЕП поділяють на *абстрактні* та *інкапсульовані*. Перші істотним чином проявляються на рівні прагматики та програмної логіки і повинні враховуватися в процесі конструювання об'єктів. Другі, навпаки, не проявляються на програмологічному рівні і повинні ігноруватися в адекватних теоретичних моделях. Програмологічний об'єкт характеризується не стільки своїми іманентними властивостями, скільки зануреністю у те чи інше середовище програмологічних відношень, що є одним з проявів *інтенсиональності* програмування.

Принцип *композиційності* уточнює поняття програмологічної операції до поняття *композиції* операції, яка з відносно простих об'єктів утворює більш складне ціле. Принцип *комутаційності* додає, що композиція виконує роль комутатора потоків даних та управління.

В подальшому будуть використовуватися такі поняття з формального апарату ЕП: аплікація – застосування функції f до аргументу d, що позначається $\mathbf{Apply}(f,d)$ або просто $f(d)$, іменна множина (ІМ), заміщення ІМ $d_1 \nabla d_2$, іменування $v \Leftarrow (d)$ та розіменування $v \Rightarrow (d)$, де параметр v – ім'я, абстрактна мультиплікація (унарна суперпозиція) $f_1 \circ f_2$, іменна мультиплікація (послідовне застосування) $f_1; f_2$, диз'юнкція (умовна композиція) $\mathbf{if}(b, f_1, f_2)$, ітерації (цикл) $\mathbf{while}(b, f)$. Означення можна уточнити за згаданими роботами.

Через NS позначимо клас ІМ, а через $is\text{–}NS$ характеристичний предикат цього класу. Нагадаємо, що теоретико-множинне об'єднання двох ІМ в загальному випадку не є ІМ.

Фундамент експлікації Рефалу

Зафіксуємо найсуттєвіші характеристики СО в цілому. За основу візьмемо думку, висловлену у [20, с. 243–244], якій надамо більш точного формулювання. Принцип *двокроковості*: загальною сутнісною ознакою логіки СО є чергування (у порядку, що визначається *механізмом управління*) етапів *аналізу аргументу* та *синтезу результату*. Аналіз аргументу уточнюється як *декомпозиція*, або *розбір*, вхідного слова. Принцип *декомпозиційної драйверованості*: рушієм обчислювального процесу у СО є процес декомпозиції слова, що обробляється.

Декомпозиція слів – розпізнавання їх структурних властивостей з одночасним виділенням складових уточнюється як *співставлення зі зразком*. Звернемо увагу на те, що декомпозиції, тобто ефективному розпізнаванню можуть підлягати лише ті структурні ознаки слова, які можливо виразити у логіці композиційних структур слів. Справді, інкапсульовані структури слів не можуть проявлятися на програмологічному рівні, а значить і на рівні розбору як основи логіки управління обчисленнями. Якщо зразки та зразкові функції як їх семантичні образи розглядати не як чорні ящики, а як об'єкти, наділені певною внутрішньою структурою, то слід прийняти принцип *дуальності*: композиції над зразками дуальні до композицій над словами.

Перейдемо від загальних характеристик СО до специфічних особливостей мови Рефал. На відміну від імперативних мов, де програми складаються з *операцій* по модифікації даних, програми у декларативних мовах, до яких належить Рефал, являють собою послідовності *правил*, які можуть застосовуватися до даних [13]. Аналогами у людській мові є для імперативних мов наказовий спосіб, а для декларативних – описовий. У людській мові "описовий спосіб є незрівнянно більш розповсюдженим, а наказовий постає у вигляді деякої спеціальної модифікації", а "відносна вага описового способу є мірою розвиненості мови" [13].

На відміну від іншої функціональної мови – Ліспу, Рефал, як і Пролог, використовує співставлення зі зразком, що дозволяє скоротити обсяг програмного тексту та зробити його більш зручним для сприйняття людиною, разом з тим, на відміну від Прологу, у Рефалі, як і у Ліспі, обчислювальний процес розгортається у прямому (від даних), а не у зворотному (від цілей) напрямку. Якщо основними структурами даних у Ліспі і Пролозі є односторонні списки, то у Рефалі двосторонні, які можуть аналізуватися і будуватися у довільному порядку, що відповідає звичайному для людини способу обробки текстів. Отже, механізми, покладені в основу Рефалу – це ті механізми, що відіграють важливу роль у людському мисленні [12].

Автори сайту [11], які мають великий досвід не лише у практичному використанні Рефалу, але й у розробці його реалізацій, характеризують нормальні алгоритми А.А.Маркова – теоретичну основу Рефалу – як "найалгоритмічнішу" з усіх універсальних алгоритмічних систем, оскільки НА в найбільшій мірі відповідають інтуїтивним уявленням про процеси переробки інформації. Логіка мови Рефал є простою і прозорою в тому сенсі, що мова не встановлює штучних бар'єрів між логікою задачі та логікою організації програмного тексту. Розвиваючи цю думку, ми могли б сказати, що при програмуванні на Рефалі акцент переноситься зі складності оформлення програмного тексту на складність розкриття внутрішньої логіки задачі, бо якщо останнє зроблено, то побудова тексту програми носить здебільшого механічний характер. Пошлемося також на автора мови В.Ф.Турчина: "структура управління Рефал-програми вельми проста та невибаглива: співставлення зі зразком та підстановка. Саме структура об'єктів, якими маніпулює програма, несе тягар зберігання корисної інформації" [10, гл. 5].

Рефал від самого початку створювався не як мова практичного програмування, а як метаалгоритмічна мова – метамова для опису синтаксису і семантики якомога ширшого класу алгоритмічних мов. При цьому, як зазначає автор мови, за мету було взято відображення характерних

для програмування засобів оперування поняттями [8, 9], зокрема засобів абстракції та конкретизації. В згаданих роботах проголошено курс на модель програмування, яка повинна відображати лише загальні риси програмування, не бути прив'язаною до якої б не було специфіки, але допускати її введення у разі потреби. Перехід від окремо взятих мов програмування до дослідження цілісної системи мов на основі метамови є прикладом метасистемного переходу, центрального поняття у загальній концепції В.Ф.Турчина [21].

Замість ролі всезагальної метамови, практика відвела Рефалу роль звичайної спеціалізованої мови з конкретною областю застосування. На нашу думку, це обумовлено рядом причин, серед яких головною є одна. Для метаалгоритмічної мови характерна абсолютизація синтаксичних аспектів програмування і спроби моделювати семантику у термінах синтаксису. Між тим, один з базових для ЕП принципів постулює підпорядкованість синтаксичного аспекту семантичному та принципову неможливість звести семантику до синтаксису.

Рефал як модифікація нормальних алгоритмів

Ряд суттєвих особливостей Рефалу зручно описати шляхом порівняння з НА [20, с. 244 і далі]. Важливою є відмінність структур слів як основного типу даних. Для НА характерна одна базова композиція над словами — конкатенація. У Рефалі крім конкатенації є композиція взяття у дужки, що незрівнянно збагачує логіку структур слів, дозволяючи утворювати похідні структури даних довільної складності. У НА для моделювання складних структур даних використовуються спеціальні символи, які розміщуються в слові, що обробляється. Хоча на рівні логіки програмування вони відіграють роль структурних маркерів, але з точки зору алгоритму обробляються як і будь-які інші символи. На нашу думку, саме можливість у явному вигляді задавати структури даних у Рефалі великою мірою забезпечує його прозорість і виразну силу.

Відмінність структур слів обумовлює відповідні відмінності структур зразків та результатних виразів. У НА допускаються зразки лише однієї стандартної структури, у термінах Рефалу *e1 S e2*, де імена *e1*, *e2* та слово *S* заздалегідь фіксовані. У Рефалі ж зразки можуть бути як завгодно складними і містити довільну кількість словарних констант та змінних. Результатні (праві) частини у формулах НА можуть мати лише вигляд *e1 T e2*, тоді як у Рефалі допускаються результатні вирази довільної складності. Отже, Рефал у порівнянні з НА характеризується пов'язаними між собою конкретизаціями одночасно у трьох напрямках: структури слів, структури зразків та структури результатних виразів. Фрагмент мови Рефал, обмежений названими трьома конкретизаціями, домовимося називати мовою Рефал-0.

НА має вигляд суцільної послідовності формул підстановки, у якій не проявляється логіка будови алгоритму. Рефал, навпаки, підтримує чітку структуризацію тексту програми — синтаксичне відокремлення відносно незалежних сукупностей речень у вигляді функцій. Конкретизувавши Рефал-0 функціональною структурою, отримаємо базисний Рефал. Дворівневий підхід, коли функціональна структура Рефалу вводиться як конкретизація лінійної послідовності речень, продемонстровано в [20]. У НА для моделювання складних конструктів управління, таких як функції, використовуються службові маркери в слові, що обробляється. Тим самим логіка управління обчислювальним процесом перекладається на дані, що спотворює структуру алгоритму. Потужність апарату функцій у Рефалі як програмологічного засобу полягає, на нашу думку, зокрема і в тому, що він дозволяє "розвантажити" дані від непритаманної їм ролі.

Композиційна структура слів

Клас $\mathcal{E}$ слів у Рефалі задається індуктивно. Порожнє слово Λ є слово. Будь-який символ s є слово (клас символів позначимо через $\mathcal{S}$). Якщо P_1 та P_2 — слова, то $P_1 \bullet P_2$ (конкатенація слів), — слово. Якщо P — слово, то (P), або *enclose* P (взяття у дужки), — теж слово. Слова вигляду (P) назвемо *дужковими*, клас дужкових слів позначимо $\mathcal{P}$. Символи та дужкові слова називають *термами*. Клас термів позначають через $\mathcal{T}$. Непорожні слова вигляду $P_1 \bullet P_2$ назвемо *конкатенаційними*, а через $\mathcal{C}$ позначимо клас таких слів. Оскільки в Рефалі поняття неуспіху має програмологічне значення, введемо спеціальний об'єкт **fail** разом з предикатом розпізнавання *is*–**fail** та будемо розглядати розширену множину слів $\mathcal{E}' = \mathcal{E} \cup \{\mathbf{fail}\}$.

Експлікація композиції конкатенації базується на властивості асоціативності

$$(P \bullet Q) \bullet R \equiv P \bullet (Q \bullet R), \qquad (1)$$

характеристичній властивості порожнього слова як одиниці відносно конкатенації

$$\Lambda \bullet P \doteq P \bullet \Lambda \doteq P, \qquad (2)$$

і на негативному твердженні про відсутність в алгебрі слів інших тотожностей. Іншими словами, конкатенація є єдиним засобом розкриття структури слів. Рівність

$$P_1 \bullet Q_1 \doteq P_2 \bullet Q_2$$

має місце тгт, коли існує таке слово P_1' або P_2', що $P_1 \doteq P_2 \bullet P_1'$ та $Q_2 = P_1' \bullet Q_2$ або $P_2 \doteq P_1 \bullet P_2'$ та $Q_1 = P_2' \bullet Q_1$. Для будь-якого слова Q та непорожнього слова P мають місце нерівності

$$Q \bullet P \neq Q,$$
$$P \bullet Q \neq Q.$$

Таким чином універсум слів, які розглядаються крізь призму конкатенаційних структур, постає як *вільна напівгрупа* відносно операції конкатенації з одиницею Λ.

Важливі наступні властивості, згідно яких конкатенаційна структура не може бути виражена через дужкову і навпаки, тобто, дані композиції повністю незалежні одна від одної:

$$enclose\ P \neq P,$$
$$P \bullet Q \neq enclose\ R.$$

Відповідно до загальних положень ЕП, композиційна структура об'єкту становить його найфундаментальніщу характеристику, відносно якої усі інші виступають як похідні, отже композиційну структуру об'єктів можливо розпізнати. Цим обумовлений вибір базових функцій.

Предикат *is–*Λ, приймає значення **true** на об'єкті Λ і значення **false** на будь-якому іншому слові. Цей предикат як *єдиний* засіб, специфічний для порожнього слова, відповідає припущенням про індивідуалізованість порожнього слова у класі слів та про його єдиність.

Єдиними суттєвими властивостями символів є їх виділеність у тому сенсі, що для кожного об'єкту можливо розпізнати, чи є він символом, а також індивідуалізованість кожного символу у тому сенсі, що для будь-яких двох символів можливо розпізнати, чи є вони одним і тим самим символом (про роль абстракції ототожнення у теорії лінійних конструктивних об'єктів див. [22, с. 35–36]). Цьому відповідають дві базові функції. Одномісний предикат $is\text{–}\mathcal{S} : \mathcal{E} \to bool$ приймає значення **true** для будь-якого об'єкту $s \in \mathcal{S}$ і значення **false** для будь-якого іншого об'єкту з класу $\mathcal{E}$. Двомісний предикат $s_1 \doteq s_2$, приймає значення **true**, коли s_1 та s_2 є одним і тим самим символом, і значення **false** в іншому випадку.

Далі, предикат *is–*$\mathcal{P}$ приймає значення **true** для будь-якого слова P з композиційною структурою $enclose\,Q$ і значення **false** для будь-якого іншого слова. Функція *disclose* здійснює розбір дужкової структури слова: якщо слово P таке, що $P = enclose\,Q$ для деякого Q, то $disclose(P) = Q$, в іншому випадку значення $disclose(P)$ не визначене. Предикат розпізнавання терму *is–*$\mathcal{T}$, який приймає значення **true** на будь-якому слові з класу $\mathcal{T}$ та значення **false** на будь-якому іншому слові, легко отримати з попередніх: *is–*$\mathcal{T}$ = *is–*$\mathcal{S}$ *or is–*$\mathcal{P}$. Аналогічно виражається предикат розпізнавання конкатенаційного слова *is–*$\mathcal{C}$.

Особливістю конкатенаційних структур є *неоднозначність* розбору. Поняття розбору конкатенаційної структури адекватно уточнюється функцією $concat^{-1}$, яка слову R ставить у відповідність *список* (див. нижче) L усіх таких ІМ вигляду $\{(1,P_i),(2,Q_i)\}$, що $R = P_i \bullet Q_i$, причому елементи списку розташовані у порядку зростання[1] слів P_i: $P_i \prec P_{i+1}$.

Крім того, є прагматичні підстави розглядати функції декомпозиції *l–hd* та *l–tl* (ліва голова та лівий хвіст), які виділяють у слові $R \in \mathcal{C}$ такі слова P, Q, $P \bullet Q = R$, що $P \in \mathcal{T}$. Іншими словами, ця пара функцій виділяє *перший* варіант декомпозиції конкатенаційного слова.

Відповідно до принципу декомпозиційної драйверованості, повинні існувати конструкти управління обчислювальним процесом, тобто композиції над функціями, що безпосередньо ґрунтуються на розпізнаванні елементарних структурних властивостей слів.

Нехай q — деяка словарнозначна функція, а p — один з предикатів розпізнавання порожнього слова, символу та терму. Підставивши у загальнозначущу композицію диз'юнкції

---

[1] За означенням, $S \prec T$ тгт, коли існує таке непорожнє слово T', що $T = S \bullet T'$, тобто коли слово S є власним початком слова T.

$\mathbf{if}(b, f_1, f_2)$ складний предикат $q \circ p$ замість предикату b, отримаємо сімейство спеціальних диз'юнкцій **if**–Λ, **if**–$\mathcal{S}$ та **if**–$\mathcal{T}$. У випадку композиції взяття у дужки приймемо означення

$$\mathbf{if}\text{–}\mathcal{P}(q, f_1, f_2) = \mathbf{if}(q \circ is\text{–}\mathcal{P}, disclose \circ f_1, f_2).$$

Спеціальну диз'юнкцію по конкатенаційній структури слова означимо як параметризовану композицію, де параметри u та v – імена:

$$\mathbf{if}\text{–}\mathcal{C}^{(u,v)}(q, f_1, f_2) = \mathbf{if}(q \circ is\text{–}\mathcal{C}, f_1', f_2),$$
$$f_1' = (u := l\text{–}hd \parallel v := l - tl); f_1.$$

Нарешті, параметризована композиція деконкатенаційного циклу $\mathbf{W}\overrightarrow{PQ}^{(u,v)}$ означується наступним чином. Якщо f – іменна функція (змістовно – тіло циклу), то функція $h = \mathbf{W}\overrightarrow{PQ}^{(u,v)}(f)$ є найменшою нерухомою точкою рекурсивного співвідношення

$$h = \mathbf{if}\text{–}\mathcal{C}^{(l,v)}(v \Rightarrow, (u := u \Rightarrow \bullet\, t \Rightarrow; h), e),$$

де e – тотожна функція. Функція h приймає на вхід слово R через ім'я v і здійснює цикл по усіх парах $\{(u, P), (v, Q)\}$, $P \bullet Q = R$, у порядку зростання слова P, до кожної такої пари застосовуючи тіло f. Це рівнозначно циклу по усіх елементах списку $concat^{-1}(R)$.

Допоміжні означення: засоби обробки списків

Суттєву роль в експлікації засобів співставлення зі зразком відіграє допоміжне поняття *списку* [5]. Клас списків об'єктів класу X позначається через $\mathfrak{L}(X)$. Композиційну структуру списків задає виділений об'єкт ε – порожній список та композиції лівого та правого приписування об'єкту до списку: якщо $x \in X$ – об'єкт, $L \in \mathfrak{L}(X)$ – довільний список, то $X \mid L$ та $L \mid X$ – теж списки з класу $\mathfrak{L}(X)$. Крім того, адекватним є залучення композиції та функції конкатенації списків, яка двом спискам L_1, $L_2 \in \mathfrak{L}(X)$ ставить у відповідність список $L_1 \Diamond L_2 \in \mathfrak{L}(X)$. Домовимося списки U-арних іменних множин називати U-арними.

Серед композицій над списковими функціями нам знадобиться композиція *спискової ітерації по успіху*, яка функції f типу $\mathcal{NS} \to \mathcal{E}'$, тобто U-арній словарній, у відповідність функцію $\overrightarrow{f \neq \mathbf{fail}}$ типу $\mathfrak{L}(\mathcal{NS}) \to \mathcal{E}'$, значення якої на аргументі L є результат першого (у порядку поелементного розбору списку зліва направо) успішного застосування функції f до елементу d списку L, або об'єкт **fail**, якщо такого d не існує, в тому числі і тоді, коли список L порожній. Іншими словами, значення $\overrightarrow{f \neq \mathbf{fail}}(L)$ задається наступним індуктивним означенням:

$$\overrightarrow{f \neq \mathbf{fail}}(\varepsilon) = \mathbf{fail},$$
$$\overrightarrow{f \neq \mathbf{fail}}(d \mid L') = f(d), \text{ якщо } f(d) \neq \mathbf{fail},$$
$$\overrightarrow{f \neq \mathbf{fail}}(d \mid L') = \overrightarrow{f \neq \mathbf{fail}}(L') \text{ в іншому випадку.}$$

Функція прямого добутку двох списків $\times^2$ ставить у відповідність спискам $L_1 = \langle x_1, \ldots, x_m \rangle$ та $L_2 = \langle y_1, \ldots, y_n \rangle$ список $L = L_1 \times^2 L_2$ вигляду

$$L = \langle \langle x_1, y_1 \rangle, \ldots, \langle x_1, y_n \rangle, \ldots, \langle x_m, y_1 \rangle, \ldots, \langle x_m, y_n \rangle \rangle.$$

Якщо $a \in X$ – фіксований об'єкт, op – двомісна операція $X^2 \to X$, то композиція $\overrightarrow{[\cdot\cdot]}$ ставить їм у відповідність функцію $\overrightarrow{[a\,op]}$ *згортки* типу $\mathfrak{L}(X) \to X$:

$$\overrightarrow{[a\,op]}\langle x_1, \ldots, x_n \rangle = a\ op\ x_1\ op \ldots op\ x_n.$$

Нарешті, композиція $\uparrow$ ставить у відповідність предикату b на X функцію $\uparrow b$ *фільтрації* типу $\mathfrak{L}(X) \to \mathfrak{L}(X)$, яка списку $L = \langle x_i \mid i = 1, \ldots, n \rangle$ ставить у відповідність список L', утворений з тих елементів x_i першого списку, які задовольняють предикатові b, у тому ж самому порядку: $L' = \langle x_i \mid b(x_i), i = 1, \ldots, n \rangle$.

Зразкові вирази у мові Рефал

Нехай κ — V-арна композиція над словами. Здійснити розбір κ-структури слова P — значить розв'язати відносно невідомої d типу V-іменної множини рівняння

$$\kappa(d) \doteq P. \tag{3}$$

Поняття співставлення зі зразком порівняно з поняттям розбору містить додаткову конкретизацію: лінійний порядок $\prec$ на множині розв'язків рівняння (3) і вимогу скінченності останньої. Семантика зразку адекватно уточнюється *зразковою функцією* функцією $\kappa_{\prec}^{-1}$ типу $\mathcal{E} \to \mathfrak{L}(\mathcal{NS})$, значенням якої на слові P є список L усіх розв'язків рівняння розбору (3), які називають *варіантами співставлення*, розташованих у порядку $\prec$. Домовимося в подальшому не виписувати індекс $\prec$.

На основі експлікації композиційної структури слів, поняття зразку у Рефалі адекватно уточнюється наступним чином: $\overline{const\,\Lambda}$ є (елементарний) зразок, який будемо називати *порожнім константним зразком*; якщо $s \in \mathcal{S}$ — символ, то $\overline{const\,s}$ є (елементарний) зразок, який будемо називати *символьним константним зразком*; якщо v — ім'я, то $\overline{symb\,v}$, $\overline{term\,v}$ та $\overline{expr\,v}$ — (елементарні) зразки типу відповідно *символьної*, *термальної* та *вербальної змінної*; нарешті, якщо C_1 та C_2 — зразки, то $enclose\,C_1$ та $C_1 \bullet C_2$ теж зразки.

Порожній константний зразок успішно співставляється лише зі словом Λ, тобто $\overline{const\,\Lambda}$ є така функція, що $\overline{const\,\Lambda}(P) = \langle\varnothing\rangle$, якщо $P \doteq \Lambda$, і $\overline{const\,\Lambda}(P) = \varepsilon$ в усіх інших випадках:

$$\overline{const\,\Lambda} = \mathbf{if}\text{–}\Lambda\left(e, \langle\varnothing\rangle, \varepsilon\right).$$

Константний зразок $\overline{const\,s}$ співставляється з одним лише символом s. Іншими словами, $\overline{const\,s}(P) = \langle\varnothing\rangle$, якщо $P \doteq s$, і $\overline{const\,s}(P) = \varepsilon$ в усіх інших випадках. Ця функція, очевидно, тісно пов'язана з предикатами розпізнавання символу $is\text{–}\mathcal{S}$ та порівняння символів $\doteq$.

Розкриваючи логічний зміст цих двох типів елементарних зразків, можна ввести поняття про *композицію константного зразку* $\overline{const}$, яка порожньому слову Λ та символу s, як об'єктам даних, ставить у відповідність функцію $\overline{const\,\Lambda}$ або $\overline{const\,s}$.

Схожим чином вводиться поняття про композицію $\overline{symb}$, яка імені v ставить у відповідність зразкову функцію $\overline{symb\,v}$, таку, що для будь-якого слова P, якщо $P \in \mathcal{S}$, тобто слово P є символом, то $\overline{symb\,v}(P) = \langle P\rangle$, в іншому випадку $\overline{symb\,v}(P) = \varepsilon$:

$$\overline{symb\,v} = \mathbf{if}\text{–}\mathcal{S}\left(e, \langle v \Leftarrow\rangle, \varepsilon\right).$$

Подібним є означення композиції $\overline{term}$, яка імені v ставить у відповідність зразкову функцію $\overline{term\,v}$, таку, що $\overline{term\,v}(P) = \langle P\rangle$ для $P \in \mathcal{T}$ і $\overline{term\,v}(P) = \varepsilon$ для усіх інших $P \in \mathcal{E}$. Композиція $\overline{expr}$ утворює таку функцію $\overline{expr\,v}$, що $\overline{expr\,v}(P) = \langle P\rangle$ для будь-якого слова P.

Опишемо тепер семантику взяття у дужки як композиції над зразками. Нехай C' — деяка зразкова функція, P — деяке слово. Тоді зразкова функція $C = enclose\,C'$ за означенням така, що $C(P) = C'(P')$, де слово P' відповідає умові $P = enclose\,P'$ (якщо таке слово існує, то воно єдине), або $C(P) = \varepsilon$ — якщо такого слова P' не існує. Внутрішній механізм та логіка цієї композиції легко розкривається за допомогою композиції $\mathbf{if}\text{–}\mathcal{P}$:

$$enclose\,C' = \mathbf{if}\text{–}\mathcal{P}\left(e, C', \varepsilon\right).$$

Співставлення з конкатенаційним зразком $C_1 \bullet C_2$ має набагато складнішу семантику:

$$(C_1 \bullet C_2)^{-1} = \left(concat^{-1} \circ \left(f \circ \cup\right)\right) \circ \left[\overline{\varepsilon\ \Diamond}\right] \circ is\text{–}\mathcal{NS}\uparrow, \tag{4}$$

$$f = \mathbf{S}^2\left(\times^2, 1 \Rightarrow \circ\, C_1^{-1}, 2 \Rightarrow \circ\, C_2^{-1}\right).$$

Іншими словами, має місце перебір усіх варіантів розбиття вихідного слова у вигляді конкатенації двох слів, перше з яких співставляється зі зразком C_1, а друге — зі зразком C_2.

На основі введених елементарних зразків та композицій над зразками можуть бути означені похідні типи зразків зі спільною структурою, що реалізують певні ідіоми програмування. По-

няття складного константного зразку означується як замикання множини елементарних константних зразків відносно композицій взяття у дужки та конкатенації:

$$\overline{const\,(enclose\,P)} = enclose\left(\overline{const}P\right),$$

$$\overline{const\,(P_1 \bullet P_2)} = \overline{const\,P_1} \bullet \overline{const\,P_2}\,.$$

Цим ми узагальнили композицію *const* на довільні об'єкти P класу $\mathcal{E}$, тобто визначили ідіому порівняння з довільним словом. Очевидна основна властивість складного зразку:

$$\overline{const\,P} = \begin{cases} \langle P \rangle & \text{якщо } Q \equiv P \\ \varepsilon & \text{в іншому випадку} \end{cases}$$

Результатні функції у мові Рефал

Особливістю результатних виразів у Рефалі є те, що вони можуть використовуватися лише разом зі зразками у складі речень, причому в результатний вираз можуть входити лише ті імена змінних, які входять у зразок з того ж речення. Зовнішні імена, значення яких присвоюються за межами речення, характерні для мови Снобол, не допускається. Це спрощує програмологічну структуру Рефалу і є одним з чинників прозорості та математичної "чистоти" мови.

На верхньому рівні абстракції семантика результатних виразів адекватно уточнюється результатними функціями (РФ) – U-арними з множиною значень $\mathcal{E}'$. Елементарними РФ є $\varnothing$-арні $\underline{const\,\Lambda}$, $\underline{const\,s}$ (де $s \in \mathcal{E}$ – параметр) та $\{v\}$-арна $\underline{var\,v}$ (де ім'я v – параметр):

$$\underline{const\,\Lambda}\,(\varnothing) = \Lambda\,,$$

$$\underline{const\,s}\,(\varnothing) = s\,,$$

$$\underline{var\,v}\,(d) = v \Rightarrow (d)\,.$$

Для РФ мають сенс композиції, обумовлені комозиціями над словами. Якщо fr, fr_1 та fr_2 – V-, V_1- та V_2-арні РФ, то $enclose\,fr$ та $fr_1 \bullet fr_2$ є V- та $V_1 \cup V_2$-арні РФ:

$$(enclose\,fr)(d) = enclose\,(fr\,(d))\,.$$

$$(fr_1 \bullet fr_2)(d) = fr_1\,(d) \bullet fr_2\,(d)\,.$$

У Рефалі при побудові результатних виразів використовуються виклики функцій f типу $\mathcal{E} \to \mathcal{E}$, що адекватно уточнюється композицією унарної суперпозиції (мультиплікації) $\circ$. Якщо результатна функція fr U-арна, то результатна функція $fr \circ f$ теж U-арна.

Замикання множини елементарних РФ відносно цих композицій дає усе багатство структур результатних виразів мови Рефал. Зокрема, легко отримати корисний з прагматичної точки зору тип складних константних результатних функцій, узагальнивши композицію *const* на довільні слова $P \in \mathcal{E}$. Функції $const\,P$ є $\varnothing$-арними:

$$\underline{const\,(enclose\,P)} = enclose\,(\underline{const\,P})\,,$$

$$\underline{const\,P_1 \bullet P_2} = \underline{const\,P_1} \bullet \underline{const\,P_2}\,,$$

$$(\underline{const\,P})(\varnothing) = P\,.$$

Речення і послідовність речень

Речення у базисному Рефалі є композиційне утворення, що складається зі зразкової та результатної функції, між якими має місце комутація по управлінню та по даних. Композицію, яка зразковій та результатній функції ставить у відповідність речення і здійснює потрібну комутацію, позначимо $\blacktriangleright$. Її семантика уточнюється співвідношенням

$$fp \blacktriangleright fr = fp \circ \overline{fr} \neq \textbf{fail}\,. \qquad (5)$$

Врахуємо тепер властивість речень керувати подальшим ходом обчислювального процесу, тобто не лише породжувати результат, але й визначати, що з ним робити далі в залежності від успіху чи неуспіху застосування речення. Нехай f_1 та f_2 – деякі словарні функції. Позначимо $Q = (fp \blacktriangleright fr)(P)$. Тоді $h = fp \blacktriangleright_{f_2}^{f_1} fr$ є словарна функція, така, що $h(P) = f_1(Q)$, якщо $Q \neq \textbf{fail}$, та $h(P) = f_2(P)$, якщо $Q = \textbf{fail}$.

Основною програмологічною конструкцією Рефалу є послідовність речень. Розглянемо її крізь призму структури списку. Через e позначимо тотожну функцію. Тоді

ε = **fail**,

$(fp \blacktriangleright fr) \mid L = fp \blacktriangleright^{e}_{L} fr$.

Зазначимо, що для побудови адекватної математичної моделі Рефалу надзвичайно велике значення має експлікація макроструктури (структури верхнього рівня) Рефал-програм, основаної на зіставленості програм з функцій. Однак це питання потребує окремого розгляду і виходить за рамки даної роботи.

Постановка задач для подальшого дослідження

Задачі, до вирішення яких підводить експлікація базисного Рефалу, можна поділити на дві групи, перша з яких відповідає підвищенню рівня конкретизації, а друга, навпаки, абстракції.

Для конструювання зразків у Рефалі допускаються елементи або гранично конкретного вигляду (константи та символьні змінні), або гранично абстрактного (термальні та словарні змінні), де за міру абстрактності приймається множина слів, що співставляються з даним елементом. Рефалу бракує засобів проміжного рівня — змінних з нетривіальною областю допустимих значень [23, гл. 3.6.1]. Серед них особливо відзначимо рекурсивні змінні, для специфікації області значень яких використовуються, в свою чергу, певні Рефал-алгоритми [23, гл. 1.4]. Еталонування зразків такого типу, безумовно, заслуговує на увагу.

Багатий спектр складних зразків дає мова Снобол [24] — інша широко відома мова СО. Логіка зразків Сноболу суттєво багатша та конкретніша за логіку зразків Рефалу [23, гл. 2.4]. У Рефалі та його розширеннях зразок *виключно* характеризується *множиною* слів, які з ним співставляються, тобто поняття зразку у Рефалі є екстенсиональним. У Сноболі ж зразок характеризується інтенсионально — з точки зору процедури співставлення. Процедура співставлення слова зі зразком у Сноболі має вищий рівень конкретизації програмологічних структур, ніж у Рефалі. Крім того, якщо у Рефалі допускаються лише локальні змінні, що набувають значень у процесі співставлення зі зразком і використовується у результатному виразі у тому ж самому реченні, то у Сноболі допускаються глобальні змінні, які можуть використовуватися поза кроком, на якому набувають значень. Сильною конкретизацією логіки програмування є і те, що у Сноболі зразки складають тип даних і отже можуть бути предметом обробки програми. Таким чином, задача еталонування програмологічних структур мови Снобол та порівняння її з відповідними структурами Рефалу нетривіальна та становить значний інтерес.

Друга група задач пов'язана з підйомом від експлікації тих чи інших конкретних мовних засобів до експлікації ідіом — прагматико-обумовлених прийомів узгодженого застосування цих засобів для вирішення тих чи інших типових прикладних задач. Іншими словами, йдеться про перехід від дрібноблочних програмологічних структур до крупноблочних. Сподіваємося, запропонованої моделі Рефалу достатньо як спільної бази для вирішення перерахованих задач.

ЛІТЕРАТУРА:

1. *Вінник В.Ю.* Про експлікацію символьної обробки: загальна характеристика проблеми та методологічні аспекти // Проблемы программирования. — 2002. — № 1-2. Спец. выпуск. Материалы 3-й межд. науч.-практ. конф. по программированию «УкрПРОГ 2002». — С. 51–57.
2. *Вінник В.Ю.* Експлікативні моделі нормальних алгоритмів // Вісник Житомирського інженерно-технологічного інституту. Серія: технічні науки. — 2002. — № 2 (21). — С. 81–87.
3. *Вінник В.Ю.* Експлікативні моделі нормальних алгоритмів // Матеріали п'ятої міської міжвузівської науково-практичної конференції викладачів, студентів та молодих вчених. — Житомир: Житомирський інженерно-технологічний інститут, 2002. — С. 18–19.
4. *Вінник В.Ю.* Еталонні моделі символьної обробки: конкатенаційний рівень // МКІМ-2002. Міжнародна конференція з індуктивного моделювання, Львів, 20–25 травня 2002: Праці в 4-х томах. — Т. 2. — Львів: Державний НДІ інформаційної інфраструктури, 2002. — С. 47–53.
5. *Вінник В.Ю.* Експлікативне програмування обробки списків // Системний аналіз та інформаційні технології. Тези доповідей учасників IV Міжнародної науково-практичної конференції студентів, аспірантів та молодих вчених (1-3 липня 2002 р. м. Київ). — К.: Політехніка, 2002. — С. 93.

6. *Вінник В.Ю.* Композиційні моделі літерних структур слів // Вісник Київського університету. Серія: фізико-математичні науки — 2002. — № 1. — С. 199–207.
7. *Вінник В.Ю.* Структури функцій символьної обробки літерного рівня // Вісник Київського університету. Серія: фізико-математичні науки — 2002. — № 2. — 10 с.
8. *Турчин В.Ф.* Метаязык для формального описания алгоритмических языков // Цифровая вычислительная техника и программирование (сборник). — М.: Сов. радио, 1966. — С. 118–124.
9. *Турчин В.Ф.* Метаалгоритмический язык // Кибернетика. — 1968. — № 4. — С. 45–54.
10. Турчин В.Ф. РЕФАЛ-5. Руководство по программированию и справочник. — http:// www. refal. net/ rf5_frm.htm
11. *Индустриальный* Рефал. — http:// case.ispras.ru/ refal/ index.html
12. *Топунов В.Л.* Язык Рефал как объект и средство обучения. — http:// ito. bitpro. ru/ 1999/ 1/ 2/ 278.html
13. Язык Рефал — http:// alice.dp.ua/ ~sergei/ lectures/ htm/ g14_2.html
14. *Мансуров В.Н.* Некоторые проблемы внешнего математического обеспечения высокопроизводительных вычислительных систем. — Материалы Всесоюзной конф. «Параллельное программирование и высокопроизводительные системы», Новосибирск, ч. 2., 1980 — С. 117–125.
15. Содружество «РЕФАЛ/Суперкомпиляция». — http:// www. refal. net/
16. *Встроенный* Рефал. — http:// case. ispras. ru/ refal/ Section_5_ Embedded_ Refal/ Embedded_Refal/ Default.htm
17. *Редько В.Н.* Основания программологии // Кибернетика и системный анализ, 2000. — № 1. — с. 35–57.
18. *Редько В. Н.* Эталонное программирование: ретроспективы и перспективы // Проблемы программирования. — 2000, № 1–2. Спец. выпуск. Материалы 2-й Межд. науч.-практ. конф. по программированию УкрПрог 2000. — с. 13–28.
19. *Редько В.Н.* Композиции программ и композиционное программирование //Программирование. — 1978. — № 5. — С. 3 — 24.
20. *Кауфман В.Ш.* Языки программирования. Концепции и принципы. — М.: Радио и связь, 1993 — 430 с.
21. Турчин В.Ф. Феномен науки. Кибернетический подход к эволюции. — http:// www.refal.net/ turchin/ phenomenon/
22. Марков *А.А.*, *Нагорный Н.М.* Теория алгорифмов. — М.: Наука, 1984. — 432 с.
23. *Головков С.Л., Наумов Н.Ф., Смирнов В.К.* О некоторых новых средствах языка рекурсивных функций, Препринт N 6 за 1982 г., М., Институт Прикладной Математики имени М.В. Келдыша АН СССР. —http:// case. ispras. ru/ refal/ Section_1_History/ Preprint_6/ Default.htm
24. *Грисуолд Р., Полонски Дж.* Язык программирования Снобол-4. — М.: Мир, 1980. — 268 с.

ВІННИК Вадим Юрійович – асистент кафедри програмного забезпечення комп'ютерної техніки Житомирського інженерно-технологічного інституту.

Наукові інтереси:

– фундаментальні проблеми теоретичної програмології, експлікативне програмування;
– формальні методи у програмуванні, семантика мов програмування;
– прикладна конструктивна математика та логіка.

Тел. (0412) 208-542

E-mail: vvinnik@ziet.zhitomir.ua; vvin@ratibor.zt.ukrtel.net

Подано 26.08.02