ТЕОРЕТИЧНІ ОСНОВИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 51.681.3

Національний університет ім. Т.Шевченко

С. Л. Кривий

Абстрактні типи даних як многоосновні алгебраїчні системи

Розглядається зображення абстрактних типів даних у вигляді многоосновних алгебраїчних систем. Детально розглядається два абстрактних типи даних: натуральні числа та списки.

Рассматривается изображение абстрактных типов данных в виде многоосновных алгебраических систем. Детально рассматривается два абстрактных типа данных: натуральные числа и списки

The image of abstract data types is examined as manysorted algebraic systems. Two abstract data types are in detail examined: natural numbers and lists.

Ключові слова: абстрактні типи даних, многоосновні алгебри, алгебраїчні системи, повнота.

1 Вступ

Абстрактні типи даних (АТД) активно використовуються в процесі розробки програмного забезпечення, його обґрунтування та верифікації. Спрощений процес розробки ПЗ можна схематично подати у такому вигляді:

Рис. 1. Спрощена схема розробки програм

За допомогою таких технологій будуються системи баз даних, баз знань, системи комп'ютерної алгебри та геометрії, графічні системи тощо. Основною вимогою при побудові таких систем є правильність та ефективність побудованих алгоритмів і програм. В процесі побудови ефективних алгоритмів і програм важливу роль відіграє вдалий вибір структур даних. Описуючи неформально алгоритм розв'язання прикладної задачі, виходячи з вибраної математичної моделі, приходиться користуватися такого типу даними, яких немає в жодній мові програмування, але які притаманні цій математичній моделі. Такого типу дані називаються абстрактними типами даних. Більш формально, під абстрактним типом даних розуміють деяку формальну математичну модель разом з операціями, функціями та предикатами, визначеними на цій моделі. Простими прикладами такого типу даних можуть служити множини, разом з операціями об'єднання, перетину та різниці. В моделі АТД оператори можуть мати операндами не тільки дані, що визначаються цим АТД, але і операнди мови програмування та операнди, визначені іншими АТД. Результатом виконання оператора теж може бути тип даних, який не визначається даним АТД. Але в рамках даного АТД припускається, що принаймні один операнд або результат довільного оператора має тип даних, визначений в даній моделі АТД. Отже, АТД логічно вписуються в поняття многоосновної алгебраїчної системи (АС) [5]. Це дає можливість визначати АТД за допомогою аксіоматики і правил маніпулювання. АТД відрізняються від структур даних, які реалізовуються в мо-

вах програмування, тим, що при використанні АТД абстрагуються від способу їх реалізації і розглядаються тільки їх властивості, які випливають із аксіоматики. Основними конструкторами, які використовуються при означенні нових операцій, функцій і предикатів є примітивна рекурсія і математична індукція. В загальному випадку означення АТД зводиться до означення таких понять:

- об'єкти,
- примітивні операції,
- конструктори нових операцій, функцій і предикатів,
- означення (дефініції) нових об'єктів.

Використання АТД дає можливість проектувати алгоритми і програми, опираючись на властивості їх операцій і предикатів з подальшим вибором найбільш ефективного способу реалізації цих операцій та предикатів. При розробці програмного забезпечення абстрагування від способу реалізації АТД має певні переваги. Ця абстракція дає можливість

– згрупувати оператори обробки АТД в одному місці програми, що значно полегшує її відладку (відлагодження);

– швидко модифікувати програмний продукт в разі зміни способу реалізації АТД (реінженерія);

– мінімізувати кількість змін в основних модулях програмного продукту, які використовують оператори обробки даного АТД.

Розглянемо в якості прикладів два типи АТД, операції та предикати, які визначені на них, а також означення нових операцій і предикатів на цих АТД за допомогою індукції та примітивної рекурсії. Всі ці АТД описуються у вигляді многоосновних алгебраїчних систем. Зауважимо, що кожну класичну алгебраїчну систему можна розглядати як двоосновну алгебру. Дійсно, якщо

$$\mathbf{A}=(\mathrm{A},\Omega)=\left\{\omega_1^{k_1},...,\omega_n^{k_n}\right\},\Pi=\left\{\pi_1^{m_1},...,\pi_r^{m_r}\right\}),\ \text{де}\ \omega_i^{k_i}:\mathrm{A}^{k_i}\to\mathrm{A},\ \pi_j:\mathrm{A}^{m_j}\to\{true,\ false\},$$
$$i=1,2,...,n,\ j=1,2,...r,$$

алгебраїчна система [5], *true* і *false* - булеві константи, то з цією АС асоціюється двохосновна алгебра

$$\boldsymbol{A}=(U=(A,\{true,\ false\}),\Omega=\left\{\omega_1^{k_1},...,\omega_n^{k_n},\pi_1^{m_1},...,\pi_r^{m_r}\right\}).$$

В цій многоосновній алгебрі кожна операція $\omega_i^{k_i}$ має тип елементів множини A, а кожний предикат $\pi_i^{m_i}$ має тип булевський. Отже, при розгляді АТД як алгебраїчної системи ми маємо справу з многоосновними алгебрами.

2 Алгебраїчна система АТД "натуральне число"

Аксіоматика. Розглянемо АТД ***N***, елементи якого називаються натуральними числами. Як відомо цей тип даних визначається аксіоматичним способом:

1. $0\in N$;

2. $s:N\to N$ - унарна операція, яка називається "наступне натуральне" і задовольняє умові: якщо $n\in N$, то $s(n)\in N$;

3. $\forall n\in N\ (s(n)\neq 0)$;

4. якщо $s(m)=s(n)$, то $m = n$.

5. Якщо A - множина така, що $0\in\mathrm{A}$ і із того, що для довільного $n\in\mathrm{A}$ випливає $s(n)\in\mathrm{A}$, то $\mathrm{A}=N$.

Це відома аксіоматична система Пеано і її алгебраїчна система "натуральне число" виглядає таким чином:

$$\mathbf{A}=(N,\Omega=\{0,s\},\Pi=\{=\}),$$

де *N* - носій алгебраїчної системи, *0, s* - нульарна та унарна операції на *N* (причому, операція $s(x)$ має вигляд $s(x)=x+1$), а = – бінарний предикат рівності, визначений на *N*.

Наявність непустих множин операцій і предикатів, визначених на множині *N*, дає можливість ввести до розгляду множини термів (алгебраїчних виразів) та формул, користуючись традиційними булевими зв'язками.

Для початку зазначимо, що кожне натуральне число можна отримати за допомогою операцій 0 і *s*. Дійсно, нехай $m\in N$ - довільне натуральне число, тоді

$$m=\underbrace{s(...s(s}_{m\ \text{разів}}(0)))...),$$

де операція *s* застосовується *m* разів до числа 0.

Аксіома 4 використовує крім операції $s(n)$ також операцію *pr(n),* яка називається "попередник" числа *n*, причому $pr(0)=0$. Ці операції

пов'язані між собою очевидними співвідношеннями $pr(s(n)) = n$ та $s(pr(n)) = n$, якщо $n \neq 0$.

Множина термів T_N визначається індуктивно.

Означення 1. Термами називаються вирази, побудовані за такими правилами:

а) 0 – терм,

б) якщо n є термом, то $s(n)$ і $pr(n)$ – терми,

в) термами є ті і тільки ті вирази, які побудовані за правилами а) – б).

Множина формул F_N визначається індуктивно аналогічним чином.

Означення 2. Формулами називаються вирази, побудовані за такими правилами:

а) $true, false$ – формули,

б) якщо m, n – терми, то $m = n$ – формула,

в) якщо A і B - формули, то $A, A \vee B, A \wedge B, A \to B$ – формули,

г) формулами є ті і тільки вирази, які побудовані за правилами а) – в).

Користуючись цими множинами, вищенаведену алгебраїчну систему можна розширити таким чином

$$\mathrm{A} = (U = (T_N, F_N), \Omega = \{0, s, pr, \neg, \vee, \wedge, \to\}, \Pi = \{true, false, =\}),$$

де $s, pr : T_N \to T_N$, $=: T_N \times T_N \to F_N$,

$\vee, \wedge, \to : F_N \times F_N \to F_N$, $\neg : F_N \to F_N$.

Отже, отримана в такий спосіб АС, є двохосновною і її називають ***моделлю Пеано***.

Використовуючи індуктивне означенням множини натуральних чисел *N*, можна ввести операцію декартового добутку $N \times N$.

Означення 3. Елементами множини $N \times N$ є ті і тільки ті елементи, які побудовані за такими правилами:

а) $(0,0) \in N \times N$,

б) якщо $(m, n) \in N \times N$, то

$(s(m), n)$ *і* $(m, s(n)) \in N \times N$.

Це означення очевидним чином узагальнюється на довільні множини $A, B, ..., C$, які мають індуктивні означення.

Приклад 1. Індуктивне означення відношення $G = \{(x, f(x)) | x \in N\}$ –"графік функції" $f : N \to N$ має вигляд

Елементами множини *G* є ті і тільки ті елементи, які побудовані за такими правилами:

а) $(0, f(0)) \in G$,

б) якщо $(m, n) \in G$, то $(s(m), f(s(m))) \in G$.

♠

Операції. Введемо операцію додавання натуральних чисел *"plus"*, користуючись операціями отриманої АС та індукцією. Це означення має вигляд:

$$plus(0, y) = y;$$

$$plus(s(x), y) = s(plus(x, y)).$$

Наприклад, обчислення значення виразу *plus*(2,1) має вигляд:

$$plus(2,1) = s(plus(1,1)) = s(s(plus(0,1))) = s(s(1))) = s(2) = 3.$$

Можна скористатися і іншим означенням операції plus:

$$plus(0, y) = y;$$

$$plus(s(x), y) = plus(x, s(y))).$$

Тоді $plus(2,2) = plus(1,3) = plus(0,4) = 4$.

Користуючись операцією *plus*, введемо операцію множення $mult(x, y)$ натуральних чисел *x* і *y*:

$$mult(0, y) = 0;$$

$$mult(s(x), y) = plus(mult(x, y), y)).$$

Приклад 2. Обчислити значення $mult(3,4)$.

$$mult(3,4) = plus(mult(2,4),4)) =$$
$$= plus(plus(mult(1,4),4),4) =$$
$$= plus(plus(plus(mult(0,4),4),4),4) =$$
$$= plus(plus(plus(0,4),4),4) =$$
$$= plus(plus(4,4),4) = plus(8,4) = 12.$$ ♠

Застосовуючи операції *s* і *pr*, введені вище операції *plus* і *mult* приймають вигляд:

$$plus(x, y) = if \ \ x = 0 \ \ \text{then } y \text{ else } s(plus(pr(x), y)),$$

$$mult(x, y) = if \ \ x = 0 \ \ \text{then } 0 \text{ else } plus(mult(pr(x), y), y).$$

З вищенаведених означень випливає, що в обох операціях використовується унарний предикат $x = 0$. Розширимо сигнатуру предикатів алгебраїчної системи цим предикатом, який позначимо $isz : N \to \{true, false\}$ (*is zero*). Тепер алгебраїчна система N приймає вигляд:

$$\mathbf{A} = (U = (T_N, F_N), \Omega = \{0, s, pr, \neg, \vee, \wedge, \to\}, \ \Pi = \{true, false, =, isz\}),$$

з множиною аксіом для введених операції $pr(x)$ і предиката $isz(x)$:

$\neg(true) = false, \ \neg(false) = true,$
$isz(0) = true; \ isz(s(x)) = false;$
$pr(0) = 0; \ pr(s(x)) = x;$
$\neg(isz(x)) \to (s(pr(x)) = x).$

В новій системі аксіоми 3 і 4 матимуть вигляд: $isz(s(x)) = false$ і $pr(s(x)) = x$ відповідно. Дійсно, з того що $s(x) = s(y)$ випливає $pr(s(x)) = x = pr(s(y)) = y.$

Зауважимо, що в цій АС предикат *eq* (предикат =) стає похідним, оскільки

$$eq(m,n) = \begin{cases} false, & якщо \quad isz(m) \wedge \neg isz(n), \\ false, & якщо \quad \neg isz(m) \wedge isz(n), \\ true, & якщо \quad isz(m) \wedge isz(n) \\ eq(pr(m), pr(n)), & інакше. \end{cases}$$

Виходячи з цього АС ***N*** приймає більш лаконічний вигляд:

$$\mathbf{A} = (U = (T_N, F_N), \Omega = \{0, s, pr, \neg, \vee, \wedge, \to\}, \Pi = \{true, false, =\}),$$

Операції *plus* і *mult* в цій АС приймають вигляд:

$$plus(m,n) = \begin{cases} n, & якщо \quad isz(m), \\ s(plus(pr(m), n)), & інакше. \end{cases}$$

$$mult(m,n) = \begin{cases} 0, & якщо \quad isz(m), \\ plus(mult(pr(m), n), n), & інакше. \end{cases}$$

Користуючись рекурсією та операціями s, pr і *plus*, введемо предикати $\leq$, $<$ та операцію зрізаної різниці $-m \div n,$ яку позначатимемо *monus* , а предикати *leq* і *less* відповідно:

$$leq(m,n) = \begin{cases} true, & якщо \quad isz(m), \\ false, & якщо \quad isz(n), \\ leq(pr(m), pr(n)), & інакше. \end{cases}$$

$$less(m,n) = \begin{cases} false, & якщо \quad isz(n), \\ true, & якщо \quad isz(m), \\ less(pr(m), pr(n)), & інакше. \end{cases}$$

Зрозуміло, що ці предикати задовольняють таким співвідношенням:

$less(0,0) = false,$ $\quad leq(0,0) = true,$
$less(s(n),0) = false,$ $\quad leq(s(n),0) = false,$
$less(0, s(n)) = true,$ $\quad leq(0, s(n)) = true,$
$less(s(m), s(n)) = less(m,n),$ $\quad leq(s(m), s(n)) = leq(m,n),$

які використовуються при побудові програм обчислення їх значень.
Означення операції зрізаної різниці приймає вигляд:

$$monus(m,n)=\begin{cases}0, & якщо \quad isz(m),\\ m, & якщо \quad isz(n),\\ monus(pr(m),pr(n)), & інакше.\end{cases}$$

Наприклад,

$monus(7,4)=monus(6,3)=monus(5,2)=monus(4,1)=monus(3,0)=3,$

$monus(3,5)=monus(2,4)=monus(1,3)=monus(0,2)=0.$

Предикати. Операція зрізаної різниці дає змогу ввести інші означення введених предикатів та нових операцій:

$m\le n=true\leftrightarrow m\div n=0,$

$m<n=true\leftrightarrow(m\div n=0\wedge\neg(m=n)),$

$|m-n\ |=plus(m\div n,n\div m).$

Використовуючи дані предикати можна визначити декартовий добуток скінченних множин. Нехай $N_p=\{0,1,...,p\}$ і $N_q=\{0,1,...,q\}$ – дві скінченні множини, тоді множина $N_p\times N_q$ визначається таким чином:

$(0,0)\in N_p\times N_q,$

$(i,s(j)),(s(i),j)\in N_p\times N_q,$

якщо $(i,j)\in N_p\times N_q\wedge s(i)\le p\wedge s(j)\le q.$

Далі, на підставі введених предикатів, вводяться тернарні функції $floor(i,m,n)=\lfloor m/n\rfloor$ (найбільше ціле i, яке не перевищує частки від ділення m на n), $ceil(i,m,n)=\lceil m/n\rceil$ (найменше ціле i, яке не менше частки від ділення m на n), а також операція $\mathrm{mod}(m,n)$ – частка від ділення m на n (перший виклик *floor* і *ceil* виконується коли $i=0$):

$$floor(i,m,n)=if\ \ less(m,n)=true\ \text{ then } i$$
$$\text{else if } eq(m,n)=true \text{ then } s(i)$$
$$\text{else } floor(s(i),m\div n,n),$$
$$ceil(i,m,n)=if\ \ less(m,n)=true\ \text{ then } s(i)$$
$$\text{else if } eq(m,n)=rtue \text{ then } i$$
$$\text{else } ceil(s(i),m\div n,n),$$
$$\mathrm{mod}(m,n)=if\ \ less(m,n)=true\ \text{ then } m$$
$$\text{else if } eq(m,n)=true \text{ then } 0$$
$$\text{else } \mathrm{mod}(m\div n,n).$$

За допомогою операції $mult$ вводиться відношення подільності натуральних чисел.

Означення 4. Натуральне число m називається кратним натуральному числу x, якщо існує таке натуральне число y, що $m=mult(x,y)$. Число x в цьому випадку називається дільником числа m (нотація $m|x$).

Властивості операцій, предикатів і відношень. Операції, предикати і відношення, означені на даній АС, мають властивості, які добре відомі.

Перш за все це предикат рівності, який є рефлексивним, симетричним, транзитивним і задовольняє властивість підстановки, тобто $\forall x,y,z\in N$

$(\mathrm{PE})\ x=x,$

$(CE)\ x=y\to y=x,$

$(\mathrm{TE})\ (x=y\wedge y=z)\to x=z,$

$(SE)\ (t=t'\wedge f(...t...))\to f(...t'...).$

Операції *plus* і *mult* задовольняють законам комутативності, асоціативності та дистрибутивності. Тобто має місце

Теорема 1.

а) $plus(x,y)=plus(y,x),$

б) $plus(x,plus(y,z))=plus(plus(x,y),z),$

в) $mult(x,y)=mult(y,x),$

г) $mult(x,mult(y,z))=mult(mult(x,y),z),$

д) $mult(x,plus(y,z))=plus(mult(x,y),mult(x,z)).$

Доведення виконаємо тільки для пунктів а) - в), залишаючи доведення решти законів читачеві.

а) (комутативність).

Покажемо спочатку, що $plus(y,0) = y$. Дійсно,

$$plus(y,0) = plus(pr(y), s(0)) = ... == plus(0, \underbrace{s(...s(s}_{y \text{ разів}}(0))...)) = \underbrace{s(...s(s}_{y \text{ разів}}(0))...) = y = plus(0, y).$$

Далі, безпосередньо із означення операції *plus* та властивостей операцій *s* і *pr* випливає, що коли $x \neq 0$, то $plus(s(y), pr(x)) = plus(y, x)$.

Тепер комутативність операції *plus* випливає із рівності таких виразів:

$$plus(x, y) = plus(pr(x), s(y)) = ... = plus(0, \underbrace{s(...s(s}_{x \text{ разів}}(y))...)) = \underbrace{s(...s(s}_{x \text{ разів}}(y))...).$$

$$plus(y, x) = plus(s(y), pr(x)) = ... == plus(\underbrace{s(...s(s}_{x \text{ разів}}(y))...), 0) = \underbrace{s(...s(s}_{x \text{ разів}}(y))...).$$

б) (асоціативність)

$$plus(x, plus(y, z)) = plus(pr(x), s(plus(y, z))) = ... = plus(0, \underbrace{s(...s(s}_{x \text{ разів}}(plus(y, z)))...)) =$$

$$= \underbrace{s(...s(}_{x \text{ разів}} plus(y, z))) = \underbrace{s(...s(s}_{x \text{ разів}}(plus(0, \underbrace{s(...s(s}_{y \text{ разів}}(z))...))))) = \underbrace{s(...s(s}_{x+y \text{ разів}}(z))...).$$

$$plus(plus(x, y), z) = plus(plus(pr(x), s(y)), z) == plus(0, \underbrace{s(...(s}_{x+y \text{ разів}}(z)))) = \underbrace{s(...(s}_{x+y \text{ разів}}(z))).$$

в) доведення аналогічне до *а)* і *б)* тільки *plus* необхідно замінити на *mult*. □

На підставі комутативності операції *mult*, отримуємо, що коли число *x* є дільником числа *m*, тобто коли існує число *y* таке, що $m = mult(x, y)$, то і число *y* теж є дільником числа *m*. Зокрема, звідси випливає, що 1 і *m* є дільниками *m*.

Відношення подільності рефлексивне, транзитивне і антисиметричне:

а) $m \mid m$ (рефлексивність),

б) $m \mid n \wedge n \mid k \rightarrow m \mid k$ (транзитивність),

в) $m \mid n \wedge n \mid m \rightarrow m = n$ (антисиметричність).

Дійсно, у випадку а) маємо:

$$mult(1, m) = plus(mult(pr(1), m), m) = plus(mult(0, m), m) = plus(0, m) = m.$$

У випадку б) на підставі теореми 1.в) і 1.г) отримуємо:

$$m = mult(n, x) \wedge n = mult(k, y) \rightarrow mult(mult(k, y), x) = mult(k, mult(y, x)).$$

Звідки отримуємо, що $m \mid k$.

У випадку в) маємо:

$m = mult(n, k) \wedge n = mult(m, s)$. Звідси отримуємо:

$$m = mult(mult(m, s), k) = mult(mult(s, m), k) =$$

$$= mult(s, mult(m, k)) =$$

$$= plus(mult(pr(s), mult(m, k)), mult(m, k)).$$

Ця рівність виконується тільки в тому випадку, коли $pr(s) = 0$ і $k = 1$, тобто $s = k = 1$.

Але тоді

$$n = mult(m,1) = mult(1, m) = plus(mult(0, m), m) =$$

$$= plus(0, m) = m.$$

На підставі властивості рефлексивності відношення подільності вводиться поняття простого числа та спільного дільника.

Число *m* називається простим, якщо воно має лише два дільники *1* і *m*.

Нехай $m, n \in N$ і існує $x \in N$ таке, що $m|x$ і $n|x$. В цьому випадку число *x* називається спільним дільником чисел *m* і *n*. Найбільше серед спільних дільників число *k* називається найбільшим спільним дільником чисел *m* і *n* (НСД*(m, n)*)

З наведених властивостей відношення подільності випливають наступні властивості:

НСД $(m, m) = m$,

НСД $(m, n) =$ НСД $(m - n, n)$, якщо $m > n$ і

НСД $(m, n) =$ НСД $(m, n - m)$, якщо $m < n$.

Звідси, як правило, починаються курси з теорії чисел та формальної арифметики, в яких ігнорується алгоритмічна підстава цієї теорії.

Приклад використання АТД. Розглянемо програми обчислення НСД двох натуральних чисел, написані в псевдокоді для цього АТД.

НСД (x, y)

Вхід: *integer x,y*.

Вихід: НСД*(x, y)*.

Метод:

1. $a := x$;
2. $b := y$;
3. *while* $a \neq b$ *do*
   3.1. if $a > b$ then $a := a - b$ else $b := b - a$
   od
4. return (*a*).

Наведений алгоритм очевидний і випливає безпосередньо з властивостей відношення подільності, що гарантує правильність алгоритму. Цей алгоритм ефективний у випадку відносно невеликих значень аргументів.

У випадку великих значень аргументів ефективнішим є так званий бінарний алгоритм обчислення НСД [3].

Бінарний- НСД (x, y)

Вхід: *integer x,y,* where $x \geq y$.

Вихід: НСД(*x, y*).

Метод:

1. $d := 1$;
2. while *x* and *y* even do
   2.1. $(x := x/2; y := y/2; d := 2d)$;
   od
3. while $x \neq 0$ do
   3.1. while x even do $x := x/2$; od
   3.2. while y even do $y := y/2$; od
   3.3. $t := |x - y|/2$;
   3.4. if $x \geq y$ then $x := t$ else $y := t$; od
4. return $(d \cdot y)$.

Цей алгоритм уже потребує доведення правильності, оскільки виконуваність вищенаведених властивостей НСД не очевидна.

В основній програмі, яка викликає програму обчислення НСД, є тільки її виклик і в разі якої-небудь зміни змінюється лише програма обчислення НСД, а не вся програма в цілому.

3. Алгебраїчна система АТД "список"

В цьому підрозділі розглядається АТД, який називається АС спискових структур. Він теж описується у вигляді многоосновної алгебраїчної системи, яка називається алгебраїчною системою спискових структур. Доводиться алгоритмічна повнота цієї алгебраїчної системи.

Алгебра спискових структур. Списки та символьні дані мають спільні означення і тому далі будемо говорити лише про списки, маючи на увазі і символьні дані. Нехай *F(X)* - вільна напівгрупа з одиницею над деяким скінченним алфавітом $X = \{x_1, x_2, ...x_n\}$. Роль одиниці відіграє пусте слово *e*. Нагадаємо, що словом в алфавіті *X* називається довільна скінченна послідовність символів цього алфавіту.

Довільне слово $p = y_1 y_2 ... y_m$ із $F(X)$ будемо називати списком елементів $y_1 y_2 ... y_m$, а самі $y_i \in X$, $i = 1,2,..., m,$ – складовими цього списку. При цьому елемент y_1 називається початком, а елемент y_m - кінцем списку. Формальне означення списка має вигляд.

Означення 5. Списком над алфавітом *X* називається множина слів *F(X),* які побудовані за такими правилами:

а) $e \in F(X)$, де *e* - пусте слово,

б) якщо $x \in X$, то $x \in F(X)$,

в) якщо $x \in X$ і $p \in F(X)$, то $x \cdot p \in F(X)$, де $\cdot : X \times F(X) \to F(X)$ - бінарна операція конкатенації символа алфавіта та слова в цьому алфавіті.

Два списки $p = s_1 s_2 ... s_m$ і $q = y_1 y_2 ... y_k$ називаються рівними, якщо $k = m$ і $s_i = y_i$ для $i = 1,2,..., m.$

З означення випливає, що *F(X)* є алгеброю з однією бінарною операцією конкатенації (•) і однією нульарною операцією (пусте слово *e*). Введемо до розгляду ще декілька предикатів і операцій над списками, тобто над елементами множини *F(X)* [1].

Нехай N - множина натуральних чисел і $p = y_1 y_2 ... y_m$ – довільне слово із *F(X)*, тоді

(1) $head(p) = y_1 \quad (head : F(X) \to F(X))$.

Іншими словами, функція *head(p)* дає перший символ слова *p*. Безпосередньо з означення цієї функції випливають такі її властивості:

$head(e) = e, \quad head(y) = y,$ якщо $y \in X,$
$head(head(p)) = head(p).$

(2) $tail(p) = y_2 ... y_m$
$(tail : F(X) \to F(X))$.

Очевидно, що
$tail(e) = e, \ tail(y) = e,$ якщо $y \in X$,
$head(p) \cdot tail(p) = p.$

Зміст наведених нижче функцій випливає з їх означення.

(3) $: F(X) \to N$ - функція довжини слова:

$$l(p) = \begin{cases} 0, & якщо \quad p = e \\ s(l(tail(p)) = 1 + l(tail(p)), & інакше. \end{cases}$$

(4) $add : F(X) \times N \times X \to F(X)$, де
$add(p, i, x) = y_1 ... y_i \ x y_{i+1} ... y_m,$
$0 \le i \le l(p).$

(5) $del : F(X) \times N \to F(X)$, де
$del(p, i) = y_1 ... y_{i-1} y_{i+1} ... y_m, \ 1 \le i \le l(p).$

(6) $hl : F(X) \times N \to F(X),$
$hl(p, i) = y_1 ... y_i, \ 0 \le i \le l(p).$

(7) $tr : F(X) \times N \to F(X)$, де
$tr(p, i) = y_{i+1} ... y_m, \ 0 \le i \le l(p).$

(8) $push : F(X) \times X \to F(X)$, де
$push(p, x) = add(p, l(p), x).$

(9) $pop : F(X) \to F(X)$, де
$pop(p) = y_1 ... y_{m-1} = del(p, l(p)).$

(10) $isemp : F(x) \to \{true, false\}$ – предикат, істинний тоді і тільки тоді, коли список є пустим:

$$isemp(p) = \begin{cases} true, & якщо \quad p = e, \\ false, & інакше. \end{cases}$$

(11) $memb : X \times F(X) \to \{false, true\}$ – предикат, істинний тоді і тільки тоді, коли елемент *x* є елементом списка *p*:

$$memb(x, p) = \begin{cases} false, & якщо \quad p = e, \\ true, & якщо \quad head(p) = x, \\ memb(x, tail(p)), & інакше. \end{cases}$$

(12) $rev : F(X) \to F(X)$ – список, елементи якого записані в зворотному порядку (конвертація списка):

$$rev(p) = \begin{cases} p, & якщо \quad p = e \vee p = x (x \in X) \\ conc(rev(tail(p)), head(p)), & інакше. \end{cases}$$

(13) $putlast : X \times F(X) \to F(X)$ – додати як останній елемент списка:

$$putlast(x, p) = \begin{cases} x, & якщо \quad p = e, \\ rev(x \cdot rev(p)), & інакше. \end{cases}$$

14) $conc : F(X) \times F(X) \to F(X)$ – конкатенація двох списків:

$$conc(p, q) = \begin{cases} p, & якщо \quad q = e, \\ conc(putlast(head(q), p), tail(q)), & інакше. \end{cases}$$

(15) $isatom : F(X) \to \{true, false\}$ – предикат, істинний тоді і тільки тоді, коли його аргумент є елементом алфавіту *X*:

$$isatom(p) = \begin{cases} true, & якщо \quad l(p) = 1, \\ false, & інакше. \end{cases}$$

Виявляється, що базовими операціями і предикатами, тобто такими, через які виражаються всі із перелічених функцій, є операції $e, \cdot, head, tail, 0, s, pr, true, false$ та оператори

суперпозиції і примітивної рекурсії. Іншими словами має місце таке просте твердження.

Теорема 2. Всі функції і предикати (3) - (15) зображуються у вигляді термів за допомогою операцій $e, \cdot, head, tail, 0, s, pr, true, false$ та операторів суперпозиції і примітивної рекурсії.

Доведення. Розглянемо послідовно випадки:

(4) $add(p,i,x) = conc(putlast(x, hl(p,i)), tr(p,i)).$

(5) $del(p,i) = conc(hl(p, pr(i)), tr(p,i)).$

(6)
$$hl(p,i) = \begin{cases} e, & якщо \quad i = 0 \\ head(p), & якщо \quad i = 1, \\ head(p) \cdot hl(tail(p), pr(i)), & інакше. \quad i > 1. \end{cases}$$

(7)
$$tr(p,i) = \begin{cases} p, & якщо \quad i = 0 \\ tail(p), & якщо \quad i = 1 \\ tr(tail(p), pr(i)), & якщо \quad i > 1. \end{cases}$$

Із зображення функцій *putlast*, *conc*, *hl* і *tr* випливає зображення функцій *add* і *del*, а також і зображення функцій *push* і *pop*.

(13) $$putlast(x, p) = \begin{cases} x, & якщо \quad p = e, \\ head(p) \cdot putlast(x, tail(p)), & інакше. \end{cases}$$

(14) $$conc(p,q) = \begin{cases} p, & якщо \quad q = e, \\ conc(putlast(head(q), p), tail(q)), & інакше. \end{cases}$$

.

Справедливість умов теореми для решти операцій і предикатів випливає безпосередньо з їх означень**.** ■

Приклад 3. Нехай $X = \{x, y, z, u, v, w\}$ і $p = xyzuv.$ Тоді маємо

1) $add(p,3,x) = hl(xyzuv,3)xtr(xyzuv,3) = head(xyzuv)hl(tail(xyzuv),2)xtr(yzuv,2) =$
$= xhead(yzuv)hl(zuv,1)xtr(zuv,1) = xyzxuv.$

2) $del(p,3) = hl(xyzuv,2)t(xyzuv,3)) = head(xyzuv)hl(tail(xyzuv),1)tr(yzuv,2) = xyuv.$

3) $hl(p,4) = head(xyzuv)hl(yzuv,3)) = xhead(yzuv)hl(zuv,2) = xyhead(zuv)hl(uv,1) = xyzu.$

4) $tr(p,3) = tail(tail(tail(xyzuv))) = tail(tail(yzuv)) = tail(zuv) = uv.$
$= (xhead(yzuv), (tail(yzuv))) = (xy, zuv).$

5) $push(p,x) = add(p, l(p), x) = add(xyzuv,5,x) = xyzuvx.$

6) $pop(p) = del(p, l(p)) = del(xyzuv,5) = xyzu.$ ♠

Слід зазначити, що функції *head* і *tail (push і pop)* є операціями на множині $F(X)$, в той час як інші є операціями многоосновними. Це дозволяє ввести таке

Означення 6. Многоосновна алгебра $G = (U = (F(X), N, \{true, false\}),$
$\Omega = \{\cdot, head, tail, e, 0, s, pr\})$ з операторами суперпозиції і примітивної рекурсії називається алгеброю списокових структур (АСС).

В цій алгебрі легко встановити справедливість таких тотожностей.

(а) $del(p,1) = tail(p) = tr(p,1);$

(б) $del(add(p,i,x), i+1) = p;$

(в) $hl(p, l(p)) = p;$

(г) $tr(p, l(p)) = e;$

(д) $pop(push(p,x)) = p.$

Має місце і ряд інших співвідношень в цій алгебрі.

Тепер, користуючись операціями і функціями (1) - (15), можна вводити і інші необхідні операції і предикати.

Як відомо, множина $F(X)$ є повністю упорядкованою відносно лексикографічного порядку, мінімальним елементом якої є пусте слово e. Користуючись цим порядком (власне цим порядком ми уже скористалися при означенні вищенаведених операцій і предикатів), визначимо за індукцією такі предикат і операцію:

а) $subword : F(X) \times F(X) \to \{false, true\}$.

Цей предикат дорівнює *true*, якщо слово *q* є підсловом слова *p*, і дорівнює *false*, в протилежному випадку. Індуктивне означення цього предиката таке: для довільного слова *q*

$$subword(p,q) = \begin{cases} false, & якщо \quad p = e \; і \; q \neq e, \\ true, & якщо \quad hl(p, l(q)) = q \\ subword(tail(p), q), & інакше. \end{cases}$$

Безпосередньо з означення випливають такі прості властивості: оскільки пусте слово є підсловом довільного слова і довільне непусте слово є підсловом самого себе, то

$subword(p,e) = subword(p,p) = true,$

$subword(p,q) = false,$ якщо $l(p) < l(q).$

б) $substit : F(X) \times F(X) \times F(X) \to F(X).$

Результатом операції *substit(p,q r)* є підстановка слова *r* замість першого входження слова *q* в слово *p*. Означення цієї функції має вигляд: для довільних слів $p, q, r \in F(X)$

$$substit(p,q,r) = \begin{cases} e, & якщо \quad p = e, \\ conc(r, tr(p, l(q))), & якщо \quad hl(p, l(q)) = q, \\ head(p) \cdot substit(tail(p), q, r), & інакше. \end{cases}$$

Безпосередньо з означення випливають такі очевидні властивості:

$substit(p,e,r) = rp, \;\; substit(p,q,q) = p.$

Приклад 4. Знайти

а) $subword(abbcda, cd)$;

б) $subword(abbc, ax)$;

в) $substit(abbcda, cd, a)$;

г) $substit(abcd, xa, da)$.

Розв'язок:

а) $subword(abbcda, cd) = subword(bbcda, cd) = subword(bcda, cd) = subword(cda, cd) = true.$

б) $subword(abbc, ax) = subword(bbc, ax) = subword(bc, ax) = subword(c, ax) = (e, ax) = false.$

в) $substit(abbcda, cd, a) = a \cdot substit(bbcda, cd, a) = ab \cdot substit(bcda, cd, a) =$
$= abb \cdot substit(cda, a) = abbaa.$

г) $substit(abcd, xa, da) = a \cdot substit(bcd, xa, da) = ab \cdot substit(cd, xa, da) =$
$= abc \cdot substit(d, xa, da) = abcd \cdot substit(e, xa, da) = abcd.$

Часто в застосуваннях необхідні структури типу "список списків". Такі структури називаються мультисписками. Цей тип даних можна означити за допомогою АТД список, якщо в якості алфавіту взяти деяку множину списків $\{p_1, p_2, ..., p_n\}$ і розглядати цю множину як алфавіт. Тоді мультисписком називаються вирази, побудовані за таким індуктивним означенням.

Означення 7. а) *e*- мультисписок,

б) $p_1, p_2, ..., p_n$ - мультисписки,

в) якщо $q_1, q_2, ..., q_k$ мультисписки, то $q_1 q_2 ... q_k$ - мультисписок,

г) інших мультисписків, крім побудованих за правилами а) -в), немає.

Як випливає з цього означення, мультисписок являє собою список, елементами якого є

списки. За допомогою мультисписків зображуються відношення в сучасних системах баз даних (відношення типу "декілька-до-декількох"), графи та інші складні математичні об'єкти.

Многоосновна АС спискових структур. Розширимо АСС предикатами рівності $=_l$ та $=_n$ для порівняння списків і натуральних чисел, відповідно. Оскільки множини операцій і предикатів непусті, то означимо множини термів T і формул F. Множина термів T складається з термів двох типів - тип цілий n (множина термів T_N) і тип списковий l (множина T_L).

Означення 8. Множина термів T складається з елементів, які побудовані за такими

а) $0, e$ - терми типу n і l, відповідно,

б) якщо $x \in X$, то x - терм типу l,

в) якщо p, q - терми типу l, а T - терм типу n, то $head(p), tail(p), x \cdot p$ і $s(m), pr(m)$ - терми типів l і n відповідно,

г) якщо $t_1,...,t_n$ - терми типів $i_1,...,i_n$ відповідно, а f - операція типу $(i_1,...i_n, j)$, то $f(t_1,...,t_n)$ - терм типу j, якщо f - побудована за допомогою операторів суперпозиції і примітивної рекурсії з термів а) - в), а $i_1,...,i_n, j \in \{l, n\}$,

д) інших термів, крім тих, які побудовані за правилами а) - г), немає.

Означення 9. Множина формул F складається з елементів, які побудовані за такими правилами:

а) $true, false$ - формули,

б) якщо $t, t' \in T_N, p, q \in T_L$, то $t =_n t'$, $p =_l q$ і $isemp(p)$ - формули,

в) якщо A, B - формули, то $\neg A, A \vee B, A \wedge B, A \to B$ - формули,

г) якщо $A_1,...,A_n$ - формули, а $P - n$ - арний предикат, то $P(A_1,...,A_n)$ - формула, якщо P побудований за допомогою операторів суперпозиції і примітивної рекурсії за правилами а) - в),

д) інших формул, крім тих, які побудовані за правилами а) - г), немає.

Виходячи з цих означень, отримуємо таку многоосновну алгебраїчну систему [4]:

$$\mathbf{A}=(U=(T_N, T_L, F), \Omega=\{0, e, \cdot, head, tail, s, pr, \neg, \vee, \wedge,\}, \ \Pi=\{true, false, isemp, =_l, =_n\})$$

з такою аксіоматикою:

а) операції та предикати з аксіоматикою для множини N,

б) $isemp(e) =_n true$,

в) $isemp(x \cdot p) =_n false$, де $x \in X, p \in T_L$,

г) $head(x \cdot p) =_l x$, де $x \in X, p \in T_L$,

д) $tail(x \cdot p) =_l p$, де $x \in X, p \in T_L$, де, $s, pr : T_N \to T_N$, $head, tail : T_L \to T_L$, $\cdot : T_L \times T_L \to T_L$, $=_l : T_L \times T_L \to \{true, false\}$, $=_n : T_N \times T_N \to \{true, false\}$, $\vee, \wedge, \to : F \times F \to F$, $\neg : F \to F$.

Отриману в такий спосіб АС будемо називати алгебраїчною системою спискових структур (АССС). Для цієї алгебраїчної системи має місце.

Теорема 3. АССС є алгоритмічно повною системою, тобто системою, в якій можна обчислити довільну частково рекурсивну функцію.

Доведення. Згідно з алгоритмічною повнотою системи нормальних алгорифмів Маркова для доведення необхідно показати, що довільний нормальний алгорифм Маркова можна виразити у цій системі.

Нехай Ф - деякий нормальний алгорифм Маркова, заданий системою формул підстановки R в алфавіті X:

$$\begin{cases} p_1 \to q_1' \\ p_2 \to q_2' \\ \ldots\ldots\ldots \\ p_m \to q_m' \end{cases}, \quad \begin{cases} p_1 \to [\cdot] q_1 \\ p_2 \to [\cdot] q_2 \\ \ldots\ldots\ldots \\ p_m \to [\cdot] q_m \end{cases}$$

де $q_i^{'} =_l \cdot q_i$ якщо формула підстановки заключна і $q_i' =_l q_i$, якщо формула підстановки проста. Далі, нехай $p \in F(X)$ і **A** - вищевизначена АССС над алфавітом $X' = X \cup \{\cdot\}$. Тоді, використовуючи операції і предикати АССС, а також функції *subword, substit* і функцію довжини, можемо записати для довільного $p \in F(X)$:

Ф(*p*) = якщо $subword(p, p_1) =_n true$ то
якщо $head(q_1') =_l "\cdot"$ то
$substit(p, p_1, tail(q_1'))$
інакше Ф $(substit(p, p_1, q_1'))$
інакше якщо
$subword(p, p_2) =_n true$ то
якщо $head(q_2') =_l "\cdot"$ то
$substit(p, p_2, tail(q_2'))$
інакше Ф $(substit(p, p_2, q_2'))$
інакше……
інакше якщо $subword(p, p_m) =_n true$ то
якщо $head(q_m') =_l "\cdot"$ то
$substit(p, p_m, tail(q_m'))$
інакше Ф $substit(p, p_m, q_m^{'}))$
інакше *p*.

Покажемо індукцією за числом *n* застосованих підстановок до слова *p*, що функція $f(p)$, яка одержана за допомогою системи *R*, і функція $f'(p)$, яка одержана за допомогою системи Ф(*p*), співпадають.

При $n = 0$ маємо $f(p) = p$, оскільки ні одна із підстановок системи *R* не застосовна до слова *p*. Із системи Ф(*p*) випливає, що $f'(p) = p$, оскільки всі умови виду $subword(p, p_i) = 0$. Отже, в цьому випадку $f(p) = f'(p)$.

Припустимо, що рівність виконується для всіх $m < n$ і нехай *n*-ою формулою підстановки є формула $p_i \to q_i'$. Можливі такі випадки: формула $p_i \to q_i'$ - заключна і формула $p_i \to q_i'$ не заключна.

Нехай в першому випадку $m = n - 1$ і після виконання *m*-ї підстановки одержане слово *p'*. За припущенням індукції $f(p') = f'(p')$. З того, що $p_i \to q_i'$ застосовна до *p'* випливає, що ні одна з підстановок які їй передують в системі *R* не застосовні до *p'*, або що те саме, що всі умови виду $subword(p', p_j) = 0$, де $head(q_i') = "."$. а умова

$subword(p', p_i) = true$ і $hrad(q_i') = "\cdot"$.

Звідси випливає, що

$$f(p') = substit(p', p_i, tail(q_i')) = substit(p', p_i, q_i) = f'(p').$$

Другий випадок аналогічний першому, з тією лише різницею, що в першому випадку обчислення зупиняються, а в другому - продовжуються. ■

Для ілюстрації роботи системи Ф(*p*) розглянемо деякі приклади.

Приклад 5. а) $X = \{a, b\}$, *R* має вигляд

$$\begin{cases} a \to \cdot e (= q_1') \\ b \to b (= q_2'). \end{cases}$$

Підставивши дані формули в систему Ф(*p*), одержуємо: Ф(*p*) = якщо $subword(p, a) =_n true$ то $substit(p, a, e)$ інакше якщо $subword(p, b) =_n true$ то Ф$(substit(p, b, b))$ інакше *p*.

Оскільки $substit(p, b, b) = p$, то даний алгорифм можна спростити:

Ф(*p* = якщо $subword(p, a) =_n true$ то $substit(p, a, e)$ інакше якщо $subword(p, b) =_n true$ то Ф(p) інакше *p*.

З одержаного виразу випливає, що Ф(*p*) не припиняє обчислень, коли непусте слово *p* не має входжень літери *a*, тобто результат застосування Ф до *p* в цьому випадку не буде визначений.

б) $X = \{x, x^{-1}, y, y^{-1}\}$, *R* має вигляд

$$\begin{cases} xx^{-1} & \to & e \\ x^{-1}x & \to & e \\ yy^{-1} & \to & e \\ y^{-1}y & \to & e. \end{cases}$$

Тоді

Ф(р) = якщо $subword(p, xx^{-1}) =_n true$ то Ф

$$(substit(p, xx^{-1}, e))$$

інакше якщо $subword(p, x^{-1}x) =_n true$ то

$$\Phi\ (substit(p, x^{-1}x, e))$$

інакше якщо $subword(p, yy^{-1}) =_n true$ то

$$\Phi\ (substit(p, yy^{-1}, e))$$

інакше якщо $subword(p, y^{-1}y) =_n true$ то

$$\Phi\ substit(p, y^{-1}y, e))$$

інакше p. ♠

На закінчення даного підрозділу зазначимо, що списки використовуються при виконанні арифметичних операцій великої розрядності в сучасних комп'ютерах.

Методи реалізації списків. Існує декілька методів реалізації АТД "список" в пам'яті комп'ютера. Вибір методу реалізації залежить від операцій, які виконуються над списками.

Реалізація на основі масивів. При реалізації списків за допомогою масивів елементи списка розміщуються в сусідніх елементах масиву. Така реалізація дає змогу швидко переглядати вміст списка і вставляти нові елементи в його кінець (операція *putlast*). Але вставка елемента в середину списка потребує переміщення всіх наступних елементів на одну позицію до кінця масиву. Вилучення елемента з середини масиву теж потребує переміщення елементів з метою заповнення пустого елемента масиву.

При такій реалізації списка визначається змінна типу запис, яка має два поля. Перше поле *elements* - це елементи масиву, чий розмір вважається достатнім для розміщення списків довільної довжини, які зустрічатимуться в даній реалізації програми. Друге поле *last* має тип цілий і вказує на позицію останнього елемента в масиві. Як видно з нижченаведеного рис.2, *i*-й елемент списка при $1 \le i \le last$ знаходиться в *i*-й чарунці масиву. Позицію елемента в списку визначає ціле число, *i*-та позиція - це просто ціле число *i*.

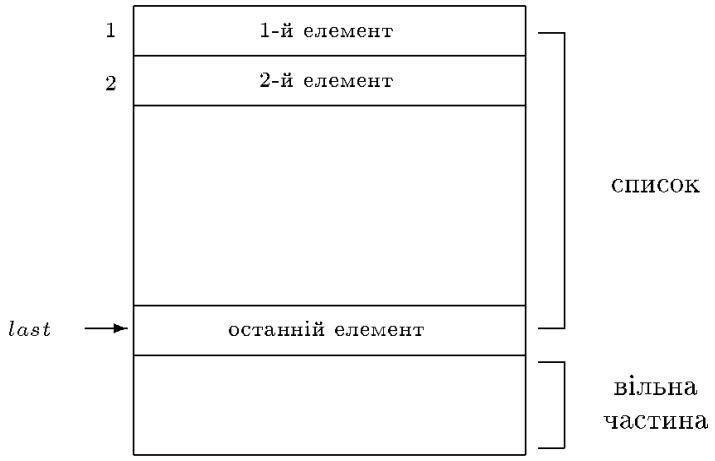

Рис 2. Реалізація списку масивом

Як випливає із сказаного, основним недоліком реалізації списка масивом є складність виконання операцій вставки і вилучення елементів. На відміну від реалізації списка масивом, реалізація списка за допомогою вказівників звільняє нас від необхідності використання неперервної області пам'яті для розміщення списку. Але при цьому необхідно використовувати додаткову пам'ять для зберігання вказівників.

Реалізація на основі вказівників. При реалізації (лінійних, однозв'язних, двозв'язних) списків використовуються вказівники, які зв'язують послідовні елементи списку. В цьому випадку список складається з елементів пам'яті, кожний з яких має два поля. Перше поле зберігає елемент списка a_i, а друге - вказівник на наступний елемент a_{i+1}. Якщо список включає елементи $a_1, a_2, ...a_n$, то елемент пам'яті в яко-

му зберігається елемент a_n має вказівник *nil* (нуль), який означає кінець списка. Першим елементом пам'яті є елемент *header* (заголовок), вказівник якого вказує на перший елемент списка. У випадку пустого списка цей вказівник є *nil.*

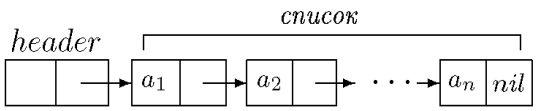

Рис.3. Реалізація списка вказівниками

Основною перевагою такої реалізації є те, що вона вимагає стільки пам'яті, скільки елементів в списку, але потребує додаткової пам'яті для вказівників елементів списку.

Реалізація на основі курсорів використовується у випадку використання мов програмування, в яких немає змінних типу вказівник (*Fortran, Algol*-60). В такому випадку вказівники моделюються цілочисельними змінними, які відіграють роль вказівників.

4 Заключне слово

На закінчення зробимо два зауваження відносно ефективності використання рекурсії та правильності алгоритмів.

Перше зауваження пов'язане з тим що в деяких мовах програмування при реалізації рекурсії виникає потреба у великих затратах пам'яті та машинного часу, а деколи і того і другого. З'ясуємо, за яких умов рекурсія виявляється кращою ніж ітерація.

Недоліки і переваги рекурсії пов'язані з тим фактом, що при попаданні на більш глибокий рівень звернення до рекурсивної програми виникає необхідність запам'ятовувати стани пам'яті та відповідні зв'язки для того, щоб мати до них доступ при поверненні на даний рівень. Якщо рекурсія занадто глибока, то для цього можуть бути задіяні великі об'єми пам'яті. Ці затрати будуть виправдані, якщо інформація, що запам'ятовується, буде використовуватися при поверненні на даний рівень. Але ці затрати можуть виявитися абсолютно непотрібними, якщо інформація не буде використовуватися [2]. Отже, не дивлячись на те, що рекурсія дає можливість будувати простим способом досить лаконічні програми, її використання може приводити до надзвичайно неефективних програм і тому необхідний певний аналіз доцільності використання рекурсії.

Зауважимо, що проблема верифікації програм і алгоритмів при використанні АТД зводиться до перевірки коректності реалізації основних операцій і предикатів відповідної АС. Наприклад, якщо розглядається реалізація АТД "список", то всі програми реалізації предикатів, функцій і операцій будуть правильними, якщо реалізація операцій $e, head, tail, \cdot$ і предикатів $true, false, isemp, =_n, =_l$ є такою, в якій істинні всі аксіоми цього АТД.

Література

1. *Кривий С. Л.* Курс дискретної математики. Київ : В-во національного авіаційного університету. - 2007. -430 с.
2. *Фостер Дж.* Обработка списков. -М. : Мир. - 1979. - 535 с.
3. *Черемушкин А. В.* Лекции по арифметическим алгоритмам в криптографии. М.: МЦНМО. - 2002. - 103 с.
4. *Hein J.L.* Discrete Mathematics. - Sudbury, Massachusetts: Jones and Bartlett Publishersю. - 1995. - 656p.
5. *Мальцев А.И.* Алгебраические системы. М.:Наука.-1970. -392 с.

Відомості про автора

Кривий Сергій Лук'янович – професор кафедри інформаційних систем Національного університету ім. Т.Шевченко, докт.фіз.-мат.наук, професор.
E-mail: krivoi@i.com.ua

Стаття надійшла до редакції 15.10.2010р.