УДК 004.415.2.043

САЛАПАТОВ В.І.

СТВОРЕННЯ ПРОГРАМ ЗА ДОПОМОГОЮ ІЄРАРХІЧОЇ АВТОМАТНОЇ МОДЕЛІ

У статті пропонується метод новий метод створення програм за допомогою ірархічної моделі кінцевого автомату. При цьому спочатку будується логічна модель програми у вигляді кінцевого автомату, а далі виконується її створення та верифікація. Стани автомату відображають виконання елементарних дій, які можуть бути описані мовою програмування високого рівня.

Ключові слова: верифікація, ієрархічна модель, кінцеві автомати, стек.

In this article new methods of creating programs by means of hierarchical model of the finite automate is proposed. On such model firstly the logical model of program, like for finite automate is created and performed its verification. States of automate reflect elementary actions which may be describes on the high level program language and then be linked into single program.

Вступ

Традиційні технології створення програм передбачають обрання алгоритму вирішення задачі, верифікації, тестування та наступного її впровадження і супроводу. Перші два етапи займають значну частину часу у процесі розробки. Тобто, спочатку програма створюється, а потім доводиться її коректність. Зазначимо, що програми являють собою певні логічні моделі, де дії, які мають виконуватись у конкретних станах моделей програм, являють собою низки елементарних інструкцій у вигляді операторів на одній з мов програмування високого рівня. В подальшому такі окремі інструкції згідно логічної моделі програми компонуються в єдиний закінчений програмний продукт. Така технологія створення програм забезпечує сумісництво етапів створення програм та їх верифікації та можливу корекцію, що значно скорочує та спрощує власно процес розробки.

Сучасні технології створення програм

Традиційні технології програмування надають сучасні мовні засоби побудови програм та їх відлагодження. Наступний етап верифікації передбачає створення моделі вже існуючої програми та її наступну верифікацію. Тобто, такий не зовсім логічний порядок виконання етапів побудови програми додає труднощів для програмістів. Дещо кращою в цьому плані є технологія UML, яка передбачає створення моделі каркасу програми та наступне заповнення усіх елементів каркасу. Але у будь-якому випадку потім виконується верифікація програми. Технологія створення програм MODEL CHTCRING прийнятна для створення розподілених та паралельних програмних комплексів з окремих модулів [2] і не вирішує проблеми створення самих модулів. Тому, більш доцільним є спочатку побудова логічної моделі програми та подальше власно її створення. При цьому стає більш простішим визначення елементарних дій, які необхідно виконати у кожному стані логічної моделі програми. За таким принципом працює комп'ютер, де кожний його стан визначає виконання певної послідовності дій, які він має здійснити. В результаті виконується програма, яка запущена в конкретний момент часу. При створенні програм також доцільно дотримуватись логічної структури алгоритму. Такий принцип забезпечує контроль правильності його виконання. Тобто, якщо коректно побудувати логічну структуру програми з самого початку, то значно спрощується етап верифікації. Логічна структура програми повинна створюватись згідно вимог технічного завдання на розробку. В процесі уточнення технічного завдання програма розбивається на рівні і на кожному рівні визначається логічна структура програми.

Побудова ієрархічної моделі програми у вигляді кінцевого автомату

Великі складні програмні проекти розробляються за принципом згори донизу. Процес розбиття великої програми на окремі частини також передбачає створення логічної структури на кожному рівні. Таким чином, створюється логічний каркас програми або модель. В подальшому на кожному рівні такий процес має по-

вторюватись. Отже модель програми в загальному вигляді можна представити наступним чином M → Mi → Mij ... → Mij...k. Тут M є верхнім рівнем моделі, а Mi, Mij, Mij...k наступні рівні у порядку спадання. На практиці таких рівнів не більше 3-4. На вищих рівнях верхівки автоматної моделі програми визначається логіка програми та часткові дії, тоді як на самому нижньому рівні верхівки автоматної моделі визначають усі необхідні дії. Такі дії на всіх рівнях можуть бути описані на будь-якій мові програмування. При цьому маємо ієрархію рівнів описів верхівок автоматної моделі програми: L → Li → Lij ... → Lij...k, де L – множини описів верхівок на всіх рівнях. Створення програми полягає у зібранні усіх описів верхівок автоматної моделі, починаючи з початкового стану по всіх рівнях і закінчуючи кінцевим станом. Оскільки модель програми створюється у процесі її розробки, то розробка програми та її верифікація здійснюються одночасно. Коректність створеної програми визначається коректністю її автоматної моделі, а також опису усіх її верхівок.

Створення програми обробки виразу

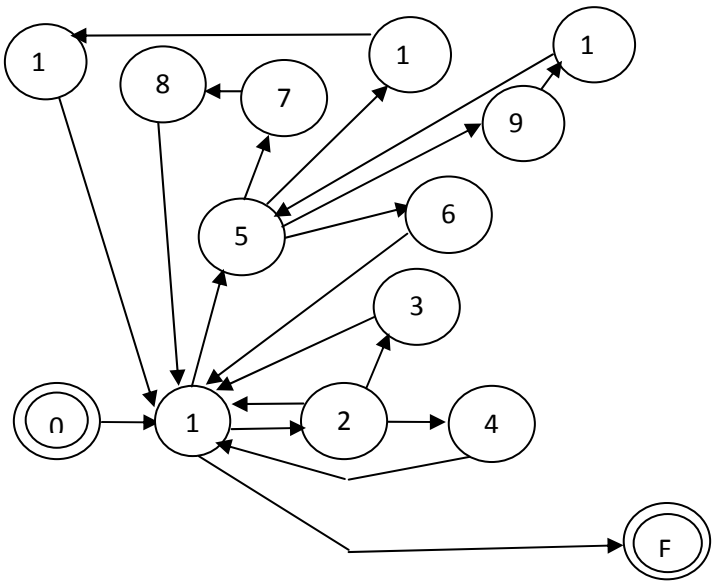
Запропоновану методику створення програм розглянемо на прикладі програми розбору виразу з використанням стекового алгоритму [2], який передбачає аналіз пріоритетності операцій. У початковому стані автоматної моделі мають бути визначені дані для програми у цілому. В даному випадку це буфер, в якому знаходиться текст виразу, буфер результату для проміжного уявлення, а також стек для розміщення в ньому пар лексем: лексем даних та лексеми дії [3]. Лексема дії складається з двох байтів – байту коду та байту приоритету. Отже, початкова верхівка визначає наступні данні.

```
VAR buff:STRING(1024);
buff1: STRING(256);
i, j, k, l : INTEGER;
CDPR  RECORD
RAZD, Pr: BYTE, VAL : INTEGER;
END;
VAR  ZAP1: CDPR;
dstack: ARRAY OF CDPR (32);
j : = 0;
```

Обробка виразу здійснюється за такими правилами.

```
 0 → 1 ; LEXAN
 1 → 2 { TERM = '' }
 2 → 3 {ZAP1.RAZD = '('};'', '('
TO_STACK
 3 → 1
 2 → 1 { ZAP1.RAZD = '+' AND
dstack(i).RAZD = beg}
 2 → 4 { ZAP1.RAZD = '-' AND
dstack(i).RAZD = beg} ;0, '-'
TO_STACK
 4 → 1
 1 → 5 { TERM < > '' }
 5 → 6 { dstack(i).Pr > ZAP1.Pr }
;TO_STACK
 6 → 1
 5 → 7 { dstack(i).Pr ≤ ZAP1.PR } ;
DOING
 7 → 8 ; FROM_STACK
 8 → 1
 5 → 9 ; { ZAP1.RAZD = ')' AND
dstack(i).RAZD # '('}; FROM_STACK
 9 → 10 { dstack(i).RAZD #
'('};DOING, FROM_STACK
 10 → 5
 5 → 11 { dstack(i).RAZD = '('};
FROM_STACK, BREAK
 11 → 12 ; SEARCRAZD
 12 → 1
 1→ F{ dstack(i).RAZD = beg AND
ZAP1.RAZD = fin}
```

Наведені правила описують переходи в автоматній моделі програми з одного стану в інший. У фігурних дужках визначені умови переходів. Якщо умови відсутні, то перехід у верхівку – безумовний. Коментарі вказують на умовні ідентифікатори, які уособлюють низку дій, що повинні виконуватись у зазначених станах. Так *LEXAN* означає дії, які можуть бути описані на нижньому рівні моделювання програми і повязані з пошуком пари лексем – лексеми даних та лексеми дії [2]. Позначки *dstack(i).Pr* та *ZAP.Pr* вказують на пріоритет попередньої та поточної дії, а їх значення beg та fin – початок стеку та кінець виразу. Позначки *dstack(i).RAZD* та *ZAP1.RAZD* вказують на коди попередньої та поточної дії. Ідентифікатори *TO_STACK* та *FROM_STACK* визначають дії у станах, які пов'язані із записом у стек пари лексем – даних та дії, а також читання їх зі стеку відповідно. Позначка *DOING* вказує на верхівку, яка пов'язана з формуванням внутрішнього уявлення частини виразу. Внутрішнє уявлення використовується як при компіляції, так і при інтерпретації. *BREAK* – вказує на верхівку 11, де знищуються ліва та права дужки. Нижче



Мал. Граф моделі програми обробки виразу

наводиться граф автоматної моделі програми обробки виразу згідно описаних правил. Позначка *SEARCRAZD* визначає пошук лексеми дії після правої дужки.

З верхівки 0 виконується перехід у верхівку 1, яка розпочинає цикл з процедури *LEXAN* та аналізу пріоритету і коду лексеми дії. Розкриємо інші введені позначки більш докладно. Верхівка 2 пов'язана з відсутністю лексеми даних та розпізнаванням при цьому припустимих лексем дії. Процедура запису в стек пари лексем даних та дії *TO_STACK*, яка виконується у верхівках 3, 4 та 6 може бути описана наступним чином:

```
dstack(i) : = ZAP1;  i : = i + 1;
```

Протилежна дія у верхівках 8, 9 та 11 *FROM_STACK* описується як:

```
ZAP1 : = dstack(i); i = : i – 1;
```

Ідентифікатор *DOING* визначає обробку формування внутрішнього подання для систем компіляції, а в разі інтерпретації – безпосереднє виконання арифметичних операцій, які обраховують даний вираз згідно коду лексеми дії. Треба зазначити, що в наведеному графі відсутній аналіз помилкових ситуацій, коли не виконується жодна з умов переходу з одного стану в інший. Вважається, що аналіз помилкових ситуацій має відбуватись у тих станах, де присутня умова переходу в інший стан. На графі також не показаний контроль сумісності типів даних. Такі дії повинні виконуватись у процедурах, які об'єднані загальним ідентифікатром *DOING*.

Для аналізу виразу, який записаний в буфері buff, розкриємо виконання дій на мові *PASCAL* в усіх станах автоматної моделі програми аналізу виразу.

Отже, для верхівки 1 маємо цикл аналізу, в якому використовується процедура *LEXAN* для виявлення пари лексем: лексеми даних та лексеми дії.

```
   k : = 0;
  L1:FOR j = k TO 256 DO
  BEGIN
       LEXAN(j);
       k : = j;
IF k = 1 THEN /*наявність лексеми
даних*/
   BEGIN
      IF  ZAP.RAZD = '('  THEN
        /* обробка лівої дужки*/
     ELSE    IF  ZAP.RAZD = '+'
  /*обробка унарного плюса*/
      ELSE   IF  ZAP.RAZD = '–'
  /*обробка унарного мінусу*/
       ELSE  /*обробка помилки*/
  END;
 ELSE
    IF    ZAP1.RAZD    =    ')'    AND
dstack(i).RAZD # '('
       END; /*обробка лексеми дії*/
    ELSE
    IF    ZAP1.RAZD    =    ')'    AND
 dstack(i).RAZD = '('  THEN
 /*BREAK; SEARCRAZD знищення дужок,
 пошук лексеми дії за правою
 дужкою*/
  ELSE    IF dstack(i).PR ≤ ZAP1.PR
THEN
  /*обробка лексеми дії*/
  ELSE    IF dstack(i).PR > ZAP1.PR
THEN
  /*запис у стек пари лексем*/
  IF   dstack(i).RAZD   =   beg   AND
ZAP1.RAZD = fin  THEN
  /*кінець обробки – верхівка F*/
```

Параметр j процедури *LEXAN* визначає позицію у символьному рядку буфера при аналізі виразу. Змінна l вказує на позицію у символьному рядку перед входом у *LEXAN*. Як було зазначено раніше [2] процедура *LEXAN* виконує посимвольний аналіз рядка і виявляє пари лексем: лексеми даних та лексеми дії. В таких парах за певних умов відсутньою може бути тільки лексема дії. На графі моделі ці умови пов'язані з діями у верхівці 2.

На даному прикладі показаний логічний каркас програми обробки виразу. Змістовне наповнення умовно позначено ідентифікаторами, які визначають певні дії. За таким принципом створена програма обробки виразу з додаванням сумісності типів даних, а також обробки вбудованих функцій на мові Pascal і в подальшому була оптимізована на мові Assembler для персональних комп'ютерів.

Висновки

Запропонована логічно зрозуміла технологія створення програмного забезпечення, яка дозволяє перед безпосереднім кодуванням програми створити її модель у вигляді кінцевого автомату. Ця модель уточнюється, корегується і перевіряється її коректність, тобто фактично частково виконується верифікація програми. Остаточна верифікація програми виконується при перевірці змісту дій у кожному стані моделі. В подальшому будується логічний скелет програми, який заповнюється змістовними операторами.

Складні програми можуть розбиватися по рівнях ієрархії, як це показано на прикладі, де на нижньому рівні подана процедура LEXAN. Оскільки процедура LEXAN вже була розглянута раніше, то в цій статті згадується тільки про її призначення. Така технологія поєднує у собі принципи структурного програмування та подання програми у вигляді моделі за допомогою розширеного недетермінованого кінцевого автомату. Така модель зрозуміло уявляє логіку роботи програми, дозволяє легко вносити необхідні корективи в разі потреби.

Список літератури

1. Ахо А., Сети А.Р., Ульман Дж. Компиляторы: принципы, технологии, инструменты. / А. Ахо, А.Р.Сети, Дж.Ульман – М.: Вильямс, 2001. – 768 с.
2. Карпов Ю.Г. MODEL CHTCRING. Верификация параллельных и распределенных программных систем. /Ю.Г. Карпов – СПб.; БХВ-Петербург, 2010.- 560 с.
3. Салапатов В.І. Синтаксичний аналіз із розподілом лексем на групи / В. Салапатов// Вісник національного технічного університету України «КПІ», Інформатика, управління та обчислювальна техніка. Київ. – 2008. – № 49. – С. 29-33.