УДК 681.325.54

ОПТИМИЗАЦИЯ АЛГОРИТМОВ ПРИ СИНТЕЗЕ ПРОГРАММНЫХ СРЕДСТВ

Д.Ю. Почекаев, И.В. Чумаченко, канд. техн. наук

Национальный аэрокосмический университет им. Н.Е.Жуковского «ХАИ»

Исследуется возможность оптимизации готовых программ, написанных на языках высокого ровня, с применением тождественных преобразований алгоритмов.

* * *

Досліджується можливість оптимізації готових програм, які написані на мовах високого рівня, з використанням тотожних перетворень алгоритмів.

* * *

The possibility of optimization of the ready programs which are written on high – level languages is explored.

В настоящее время размеры программных продуктов и фирм-производителей программ мирового уровня и программистов-одиночек чрезвычайно велики. Примером тому является всё возрастающие требования к объёму памяти операционных систем Windows – комфортная работа в Windows 95 обеспечивается при наличии в компьютере 16 МБ ОЗУ, в Windows 98 для комфортной работы необходимо, как минимум, 32 МБ ОЗУ, Windows 2000 при инсталляции проверяет компьютер на наличие в нём 64 МБ ОЗУ. Это говорит о постоянно растущем размере программных продуктов. Это характерно не только для программ фирмы Microsoft, но и для продуктов других фирм: например, каждая новая версия популярного броузера Netscape Communicator требует всё больше и больше места на жёстком диске и в ОЗУ. Инсталлятор Netscape Navigator 3.0 требует на диске менее 5,6 МБ, объём инсталлятора Netscape Communicator 4.04 – 17,2 МБ. Очевидно, что чем больше инсталлятор, тем больше и сам программный продукт. Этот рост требований к ОЗУ свидетельствует о постоянном росте размеров исходных текстов программ. Налицо множество факторов, обуславливающих неотвратимый рост размеров программных продуктов от версии к версии. Это и исправление обнаруженных ошибок, и создание реакции программы на ранее непредусмотренные воздействия пользователя, и доработка оформления интерфейса и, естественно, введение новых возможностей, нереализованных по разным причинам в предыдущей версии программы. Чем больше размер программы, тем сложнее в ней ориентироваться, обновления, дописки и исправления ошибок наслаиваются, программист перестаёт понимать назначения не только конкретных строк программного кода, но и целых процедур. Это положение усугубляется тем, что над программным продуктом работает, как правило, группа программистов. Результатом нагромождения команд в исходном коде является появление двойных и тройных проверок одних и тех же условий, дублирование алгоритмов обработки данных и т.п. следствием всего этого является замедление работы программы и завышенные требования к ОЗУ.

Для того, чтобы снизить негативные последствия многократных исправлений программного кода, фирма Netscape заявила о том, что новая версия её броузера Netscape Communicator 6.0 не содержит ни одной строки программного кода из предыдущих версий. Однако подобный подход не является универсальным методом создания оптимальных по размеру и быстродействию программ больших объемов. Для уменьшения размера программы, а как следствие, и повышения её быстро-

действия, можно с помощью формальных преобразований её алгоритма добиться исключения всех повторяющихся или неиспользуемых операторов, не меняя алгоритма работы программы и не влияя на конечный итог её работы.

Чтобы применять этот метод на практике, необходимо получить формулы для преобразования и доказать, что применение этих формул не искажает результатов работы оптимизируемой части алгоритма или алгоритма в целом. За основу возьмём комплекс тождеств систем алгоритмических алгебр (САА), предложенных В. М. Глушковым. [1,2]

1. QE = EQ = Q;
2. P v P = P;
3. (PQ)R = P(QR);
4. (P v Q) v R = P v (Q v R);
5. P v Q = Q v P;
6. P (Q v S) = PQ v PS;
   (P v Q) S = PS v QS;
7. {{P}} = {P};
8. {P} = E v P{P};
9. P{P} = {P}P;
10. {P}{P} = {P};
11. {P} v P = {P};
12. {E} = E,

где Q, R, P – произвольные операторы,

R·Q – запись последовательного выполнения операторов R, Q,

{…} – цикл с произвольными условиями;

Q v R – запись альтернатив выполнения операторов R и Q,

E – событие, состоящее из одного пустого слова.

Список тождеств 1÷12 является базовым, на его основе можно вывести более сложные формулы для формального преобразования алгоритма программы, представленной в виде последовательности литер операторов.

Список предложенных тождеств может быть продолжен. Добавим понятие фильтра:

13. $\underline{\alpha} = (E \vee N)$,

где α – логическая операция,

N – неопределённый оператор, такой, что:

14. NQ = QN = N.

Выведем несколько тождеств на основании перечисленных ранее для формальных преобразований на базе фильтра:

15. $\underline{\alpha}P = P\,{}_{\alpha}(E \vee N)$.

16. ${}_{\alpha}\{R\} = \bar{\underline{\alpha}} \vee \underline{\alpha} R\,{}_{\alpha}\{R\}$.

17. $\{S\}R = \{S\}R \vee R$.

18. ${}_{\bar{\alpha}}(P \vee R) = {}_{\alpha}(R \vee P)$.

Для применения этих тождеств следует преобразовать текст программы в последовательность литер, обозначающих операторы исходной программы.[3,4,5] Затем надо просматривать полученную последовательность литер на предмет применения к ней разработанных тождеств, подобно применению в алгебре формул сокращённого умножения. Особый интерес при преобразовании представляют литеры, обозначающие такие ключевые слова языка, как **If, Then, and, not** и т.п.

Чтобы доказать возможность подобных преобразований без искажений алгоритма работы программы, воспользуемся предложенным набором правил на примере, применив их к программе текстового редактора, написанной на языке Delphi.

```
...
j := Length(RichEdit1.Lines[i]);
If not (j > 0)
Then
  Leng := j
Else
begin
  Repeat
    s := RichEdit1.Lines[i];
```

```
        Ch := s[j]; // отбрасывание
пробелов с конца
        j := j - 1;
      Until (j = 0) or (Ch <> ' ');
      Leng := j;
    end;
    ...
```

Просматривая последовательно программу, отыскиваем часть кода, к которой можно применить правую часть тождеств. После применения тождеств 17 и 18 получим следующий результат.

```
    ...
    j := Length(RichEdit1.Lines[i]);
    If (j > 0)
    Then
    begin
      Repeat
        s := RichEdit1.Lines[i];
        Ch := s[j]; // отбрасывание
пробелов с конца
        j := j - 1;
      Until (j = 0) or (Ch <> ' ');
    end;
    Leng := j;
    ...
```

Использовав тождества 17 и 18 нам удалось из программы исключить **Else -** часть оператора **If** и использовать строку `Leng := j;` один раз, поместив её после оператора **If**.

Заключение

Из вышеизложенного можно сделать следующие выводы: расширив набор тождеств и применяя их для больших программ можно получить заметное сокращение размера (ок. 5%) исходного кода программ, а самое главное –упрощается сам алгоритм программы, что приводит к повышению быстродействия программного продукта.

Литература

1. В. Я. Жихарев, В. М. Илюшко, Н. В. Нечипорук, И. В. Чумаченко «Методы проектирования символьных процессоров» – Харьков, «Факт», 2000 г. 183 с.

2. В. М. Глушков «Синтез цифровых автоматов» – М., Государственное издательство физико-математической литературы, 1962 г. 90 с.

3. Ершов А. П. «Введение в теоретическое программирование» – М., Наука, 1997 г. – 289 с.

4. Жихарев В. Я., Илюшко В. М., Чумаченко И. В. Проектирование электронных компиляторов: Моногорафия. – Харьков, «Факт», 1999 г. 88с.

5. Жихарев В. Я., Чумаченко И. В. Синтез символьных процессоров // Оптимизация управления, информационные системы и компьютерные технологии: Труды Украинской академии экономической кибернетики (Южный научный центр). Киев-Одесса: ИСЦ, 1999. – Вып. 1. Ч. 1. – С. 63-70.