К ВОПРОСУ О КОРРЕКТНОСТИ ПРОГРАММ

Нана Бенидзе

Сухумский филиал Тбилисского государственного университета им. Ив.Джавахишвили
ул. Джикиа, 12, Тбилиси, 380000, Грузия, тел.: (99593) 24-67-43 E-mail: n_benidze@yahoo.com

Аннотация

В мире программирования, проблема корректности или истинности тех или иных программ (алгоритмов) всегда была актуальна. После того, как объектно-ориентированная технология стала «жемчужиной» программирования и можно разработать довольно серьезные программы, это проблема достигла немыслимых масштабов.

Чтобы доказать истинность конкретных алгоритмов, легче всего использовать тестирование. Метод тестирования так же стар, как наш мир. Если, для большего множества начальных данных программа правильно работает, то вероятность того, что эта программа корректна, довольна велика. Но если, та же программа, хотя бы для одного набора начальных данных работает не правильно, то все результаты тестирования пропадут в пустую.

Цель этой статьи сравнение двух методов, чтобы доказать корректность программ: 1) с помощью правил Консеквенции и 2) с помощью конечного автомата.

Можно предложить два подхода:

1. Надо придумать строго определенный математически механизм, с помощью которого можно доказать истинность любого алгоритма.
2. Надо построит конечный автомат, после работы которого, за конечный период времени, докажем корректность любой программы.

Чтобы доказать корректность программы с помощью первого подхода, достаточно:

1. Произвести декомпозицию начальной программы на конечное число подпрограмм и если сможем доказать истинность каждой подпрограммы, то тогда корректность первоначальной программы будет доказана.
2. Повторить этот процесс для каждой подпрограммы.
3. Повторить 1-2 до достижении таких подпрограмм, корректность которых можно доказать без каких либо осложнении.

Идея достаточно проста и ясна.

Допустим, S – программа, правильность которой надо доказать, P – отношение, которое определено на начальные данные, т.е. предусловие, а Q – отношение определенное над конечными значениями, т.е. постусловие. Тогда имеет место следующая формула:

$$\{ P\} \ S \ \{ Q \} \qquad (1)$$

(1) означает следующее: если P предусловие истинно перед выполнением S программы и S программа корректно, то Q постусловие будет истинным после завершения выполнения программы S.

Например:

$\{$ A массив, из целых чисел $\}$ SORT(N,A) $\{ A_i < A_i+1, i<=N\}$

где:

$P \cong \{$ A массив, из целых чисел $\}$

$S \cong$ SORT(N,A)

$Q \cong \{ A_i < A_{i+1}, i<=N\}$

если $S \cong$ SORT(N,A) алгоритм правильный, тогда исходя из P, условие $\{ A_i < A_{i+1}, i<=N\}$ должно быть истинным

В этих обозначениях первый подход можно сформулировать так:

1. программу S можно рассмотреть как совокупность S_i (i=1..n) подпрограмм (или фрагментов), соответственно P_i предусловие, а Q_i постусловие для каждой S_i

   $$\{ P_i \} S_i \{ Q_i \}$$

   если сможем доказать истинность каждой S_i, то тогда та программа, которая строится из этих S_i блоков, будет истинным.
2. повторить первый шаг для каждой S_i.
3. повторить 1-2 шага до достижений таких Si , корректность которых можно доказать за просто.

(Введем обозначение:

$$\frac{D1, D2, ..., Dn}{D},$$

означает, что если D1, D2 , ..., Dn корректные утверждения, то D является истинным).

Аксиома 1:

$$\frac{\{P\}\ S\{R\} \quad u \quad R \Rightarrow Q}{\{P\}\ S\{Q\}}$$

если программа S обеспечивает истинность утверждения R, то она также обеспечивает истинность каждого утверждения, которое следует из R.

Аксиома 2:

$$\frac{P \Rightarrow Q \quad \{R\}\ S\{Q\}}{\{P\}\ S\{Q\}}$$

если R известное предусловие программы S, приводящей к результату Q, после завершения его выполнения. То это же относится к любому другому утверждению из которого следует R.

1-2 аксиомы называются правилами *Консеквенции* [1].

Из правил Консеквенции вытекает следующие правила:

3) правила для пустого оператора:

$$\{ P\} ; \{ Q \}$$

4) правила для составного оператора:

$$\frac{\{Pi-1\}\ Si\{Pi\}, i=1..n}{\{P0\}\ begin\ S1, S2, \ldots Sn, end\ \{Pn\}}$$

5) правила для условного оператора:

5.1 $$\frac{\{P \& B\}\ S1\{Q\}, \{P \& \sim B\}\ S2\{Q\}}{\{P\}\ if\ B\ then\ S1\ else\ S2\ \{Q\}}$$

5.2 $$\frac{\{P \& B\}\ S\ \{Q\}, P \& \sim B \Rightarrow Q}{\{P\}\ if\ B\ then\ S\ \{Q\}}$$

6) правила для оператора выбора:

$$\frac{\{P \& (x = k1) Si\{Q\}, i = 1..n}{\{P \& (x \in [k1, \ldots, kn]) caseof k1 : S1; \ldots; kn : Sn; end\{Q\}}$$

7) правила для оператора while – do:

$$\frac{\{P \& B\}\ S\ \{P \& \sim B\}}{\{P \& B\}\ while\ B\ do\ S\ \{P \& \sim B\}}$$

8) правила для оператора repeat – until:

$$\frac{\{P\}\ S\ \{Q\},\ Q \& \sim B \Rightarrow P}{\{P\}\ repeat\ S\ until\ B\ \{Q \& B\}}$$

$$\frac{\{P1\}\ S\{Q\}, \{P2\}\ S\{Q\}}{\{P1 \vee P2\}\ S\{Q\}}$$

Опираясь на этих правилах можно доказать корректность любого алгоритма. Например, докажем истинность алгоритма Эвклида – нахождении наибольшего общего делителя двух натуральных чисел.

Текст программы:

```
        . . .
int a,b,x,y;
        cin>>a>>b;
        while(a!=b)                                                    (10)
           if(a>b)  a-=b;
           else b-=a;
        cout<<" наибольший общи делитель"<<x<<"  и  "<<y<<" равен "<<a;
        . . .
```

Наша задача: с помощью правил консеквенции и вытекающих из них правил доказать истинность (10) программы.

Введем обозначение:

P ≅ {a>=0 & b>=0 & a==b} // предусловие

Тогда 3-е правило гласит, что (10) программа истинна, когда P имеет вышеуказанный вид.

Рассмотрим другой вариант P предусловия:

P ≅ {a>=0 & b>=0 & a!=b}

```
{a>=0 & b>=0 & a!=b}        //  предусловие
                while (a!=b)
                        if(a>b)  a-=b;                                  (11)
                          else b-=a;
            {a>=0 & b>=0} &{a==b}     //     постусловие
```

если обозначим:

```
S ≅   if(a>b)  a-=b;                                                    (12)
        else b-=a;
```

P ≅ {a>=0 & b>=0}

B ≅ { a!=b}

тогда с помощью 7-го правила (11) истинна.

Осталось доказать истинность S. Если обозначим:

S1 ≅ a-=b;

S2 ≅ b-=a;

B ≅ {a>b}

P ≅ {a>=0 & b>=0}

Тогда с помощью 5-го правила:

```
{a>=0 & b>=0}
  if B then S1
        else S2;
{a>=0 & b>=0}
```

т. е. (12) истинна.

ч. т. д.

Ясно, что мы имеем дело с, не так уж сложным алгоритмом. Теперь обсудим более сложную задачу – синтаксический разбор арифметического выражения.

Приведем синтаксическую формулу арифметического выражения (используем формализм Бекус - Наура):

```
<арифметическое выражение>::=[+|-] <Терм>{+|-<Терм>}
<Терм>::= <множитель> {* | / | % <множитель>}                           (13)
<множитель>::= <число> | < идентификатор> | (<  выражение  > )
```

(13) синтаксические формулы сами определяют семантику соответствующей программы.

Текст программы:

```
                        . . .
 #define probel while(c==' ') c=getchar();
  void Expression();
  void Term();
  void mult();
  void number();
  void ident();
    main()
   {  char c;
      c=getchar();  probel
      Expression(); probel                                              (14)
  if(c!=EOF) {cout<<"Syntax Error";
              exit(0);}
         cout<<"Translation complete";
   }
                        . . .
void Expression()
{  if(c=='+' || c=='-') {c=getchar(); probel}
        Term(); probel                                                  (14.1)
    while(c=='+' || c=='-')
      {Term(); probel}
}
                        . . .
void Term()
{   mult();
```

```
            while(c=='*' || c=='/')                                          (14.2)
        {mult(); probel}
}
                        . . .
void mult()
{  if(c>='0' && c<='9')  number();
      else if(c>='a' && c<='z' ||  c>='A' && c<='Z' || c=='_')  ident();
           else if (c=='(')
                {Expression(); probel                                        (14.3)
                   if( c!=')' ){cout<<"Syntax Error";
                                exit(0);}
                   }
          c=getchar();
}
                        . . .
  void number()
{  while((c=getchar())>='0'&&c<='9');                                        (14.4)
}
                        . . .
void ident()
{  while((c=getchar())>='a'&&c<='z' || c>='A'&&c<='Z' || c>='0'&&c<='9' || c=='_');   (14.5)
}
```

Можно сказать, что программа достаточно прозрачна, но имеет одну серезьную сложность: алгоритм рекурсивен и имеет место косвенная рекурсия.

Если докажем, что функция Expression() истинна, то тогда ясно, что (14) программа истинна (макрос probel можно игнорировать).

Введем обозначения:

```
S1 ≅ if(c=='+'|| c=='-') c=getchar();
S2 ≅ Term();                                                                 (15)
S3 ≅ while(c=='+' || c=='-')
                Term();
```

S1 программа истинна в силу 5.2. Если сможем доказать истинность S2 программы, тогда в силу 7-го правила S3 программа будет истинной.

Т.е. чтобы доказать истинность (14.1), надо доказать истинность (14.2). По аналогию (15) схемы, в конце концов надо доказать истинность (14.3) – это то же самое, что доказать истинность функции mult().

Введем обозначения:

```
S1 ≅ if(c>='0'|| c<='9') number();
S2 ≅ if(c>='a' && c<='z' ||  c>='A' && c<='Z' || c=='_')  ident();
S3 ≅ {Expression(); probel                                                   (14.3)
            if( c!=')' ){cout<<"Syntax Error";
                         exit(0);}
            }
```

Интереснее всего S3 программа: для того чтобы доказать истинность функции mult(), надо доказать истинность функции Expression(). Т. е. вошли в тупик – по иронии судьбы, это дорога не ведет в «Рим».

С помощью аксиом Консеквенции, в этот раз не удалось доказать истинность (14), потому, что она рекурсивная программа: доказать истинность функции Expression() это то же самое, что доказать истинность mult() функции, с другой стороны, чтобы доказать истинность mult() функции, надо доказать истинность Expression() функции.

Теперь попытаемся доказать корректность (14) с помощью конечного автомата.

Идея, как и в первом подходе проста: каждый прочитанный символ попадает на конечный автомат один раз, и обработка этого символа занимает конечный период времени. Так что общее время работы конечного автомата не превосходит $\theta(n)$ (где n длина алфавита – т.е. количество символов, с помощью которых можно записать арифметическое выражение: 27 больших букв латинского алфавита, 26 прописных букв, 10 цифровых символов, 5 арифметических операции).

Но все это справедливо, если такой автомат уже существует, т.е. осталось самое «малое», построить конечный автомат для синтаксического разбора арифметического выражения.

По определению, конечный автомат представляет собой [2]:

$$F = \{Q, q_0, A, \sum, \sigma\},$$

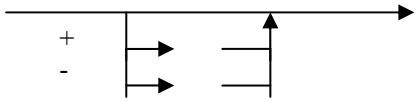
где Q – конечное множество состоянии автомата; $q_0 \in Q$ – начальное состояние автомата;
$A \subseteq Q$ – конечное множество возможных состоянии автомата; $\sum$ – конечный начальный алфавит;
σ – функция из $Q \times \sum$ в Q, и она называется функцией переходов.

С самого начала автомат находится в состоянии q_0. Потом он по очереди читает символы из входного файла (арифметическое выражение при синтаксическом разборе находится в файле). Когда автомат стоит в q состоянии, читая очередной С символ, он переходит в новое σ (q, C) состояние. Если σ (q, C) $\in$ A, то тогда говорят, что прочитанный символ принят конечным автоматом, если σ (q, C) $\notin$ A, то говорят, что переход не осуществился и для (13) программы, если она корректна, естественно, это означает, что произошла синтаксическая ошибка или же в худшем случае программа не корректна.

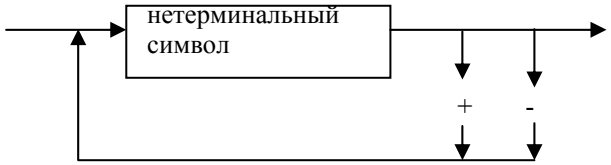
Задача поставлена, осталось главное – построить конечный автомат.

Из за того, что разбор арифметического выражения немного сложная задача, поэтому схема соответствующего конечного автомата графический выглядит как то непривычна.

Введем несколько схематических обозначении:



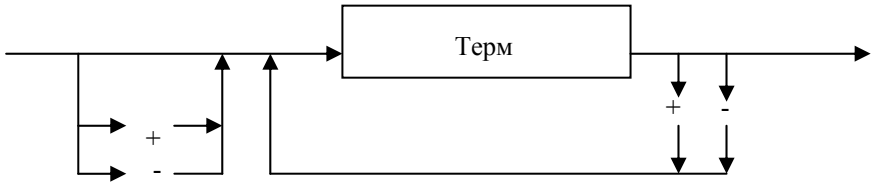
обозначает, очередной символ или символ +, или -, или не одно, не другое.



повторение

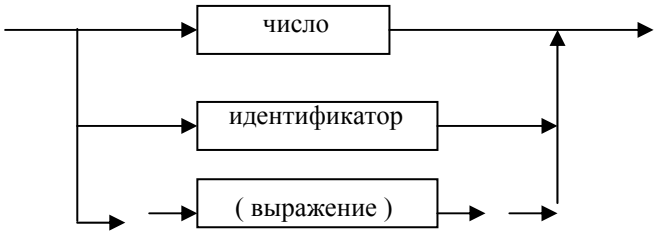
Кто знаком с синтаксическими диаграммами для них эти обозначения естественный. В силу этих обозначении, имеет место следующие псевдоавтомати.

Псевдоавтомат, который соответствует арифметическому выражению:

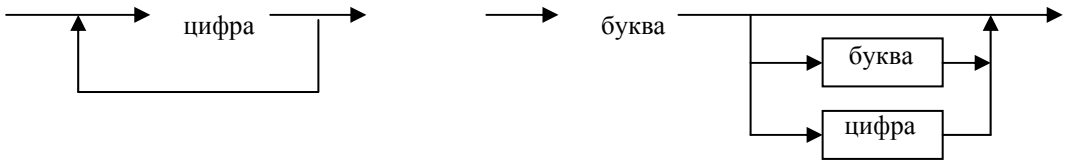


Псевдоавтомат, соответствующий Терму:

Псевдоавтомат для множителя:



Псевдоавтомат, соответствующий числу и идентификатору:



С помощью этих псевдоавтоматов, заключительный вид конечного автомата, который соответствует арифметическому выражению, представлен на рис. 1.
где ∑ = {'a'..'z', 'A'..'Z', '0'..'9', '+' '-', '/', '%', '*', '_'}; σ = {Expression, Term, mult, number, ident},
в роли σ из Q X ∑ в Q, может быть из перечисленных функции только один в данный момент времени.

Множество Q, исходя из начальной задачи, больше похоже на графа, но ясность возможных состояний все же сохраняется.

qnext = σ (qpre, C) $\notin$ A множеству, это означает, что переход с помощью автомата не осуществился не на одном направлении и синтаксический анализ закончился с ошибкой, или (13) не истинна.

Если все qnext $\in$ A и с помощью конечного автомата смогли дойти до qfinal, это значит что синтаксический анализ закончился удачно.

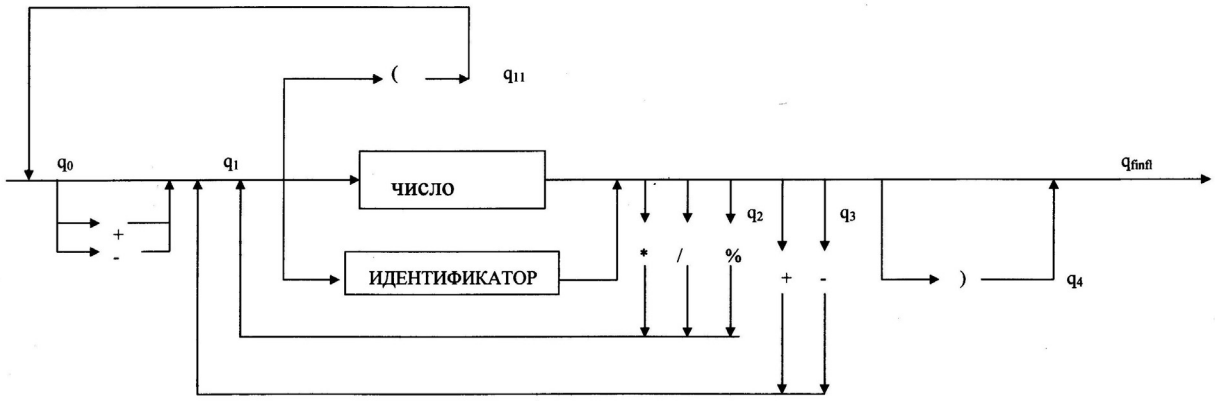


Рис. 1.

Заключение

Цель этой статьи сравнение двух методов, чтобы доказать корректность программ: 1) с помощью правил Консеквенции и 2) с помощью конечного автомата. И выбрать одно из них, как более удачный.

Но этого не удалось. Если корректность не рекурсивного, вычислительного алгоритма смогли доказать без каких либо сложностей с помощью правил Консеквенции, то это не удалось по отношению рекурсивного алгоритма. В то время как, истинность того же алгоритма с помощью конечного автомата можно доказать достаточно проста.

Можно предположить, что композиция этих двух подходов дает возможность доказать истинность любого алгоритма: рекурсивного и не рекурсивного.

Литература:

[1] Алагич С., Арбиб М. "Проектирование корректных структурированных программ." Пер. с англ. М. Радио и связь 1984.
[2] Ахо А., Ульман Дж. "Теория синтаксического анализа, перевода и компиляции (Том 1. Синтаксический анализ)" издательство «Мир» Москва 1978