Динамический анализ информационных зависимостей в цикле (название проекта: RuntimePrecheck)

Основная идея преобразования

Преобразование *RuntimePrecheck* предназначено для решения проблемы статически неопределяемых информационных зависимостей в цикле типа *for*. Такие информационные зависимости возникают между вхождениями массива (далее анализируемый массив), индексные выражения которых содержат разного рода «аномалии»: косвенная адресация, нелинейные выражения от счетчика цикла, переменные.

Данное преобразование осуществляет предварительный анализ информационных зависимостей на этапе выполнения программы. Для этого в указанном месте программы создается специальный цикл, который фиксирует все обращения к анализируемому массиву. Далее накопленная информация подвергается анализу. В результате генерируется признак, который сигнализирует о наличии или отсутствии определенных видов информационных зависимостей.

Требования, предъявляемые к входным данным

На вход функции подается ссылка на преобразуемый цикл и ссылка на оператор, перед которым или после необходимо вставить фрагмент преобразования.

Анализируемый цикл и место, где требуется осуществить тестирование, должны принадлежать общему пространству имен, так чтобы все переменные доступные из цикла были доступны и в данном месте программы.

К исходному циклу предъявляются следующие требования:

1. Цикл типа for;
2. Индекс (счетчик цикла) не меняется в теле цикла;
3. Вхождения индексного массива и вхождения анализируемого массива не связаны информационными зависимостями (проверять до тех пор, пока не встретим константу или не достигнем заголовка цикла);
4. Содержание индексных массивов не меняется в теле цикла.

О проверочном цикле

Проверочный цикл создается с пустым телом. Заголовок цикла такой же, как и у исходного. Необходимые операторы, в том числе управляющие конструкции, добавляются в естественном порядке.

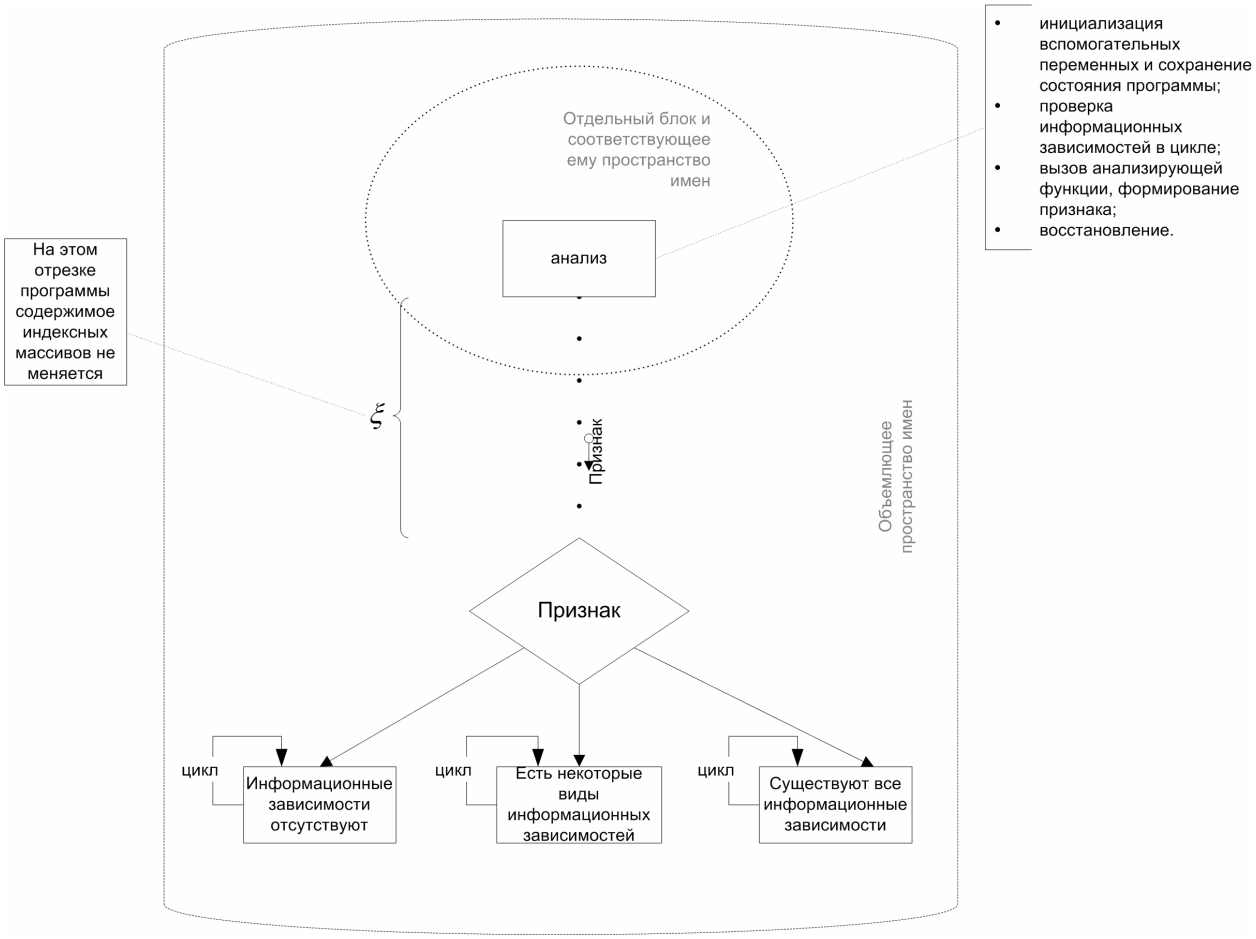
Переменные, входящие в выражения-условия управляющих конструкций, должны удовлетворять следующим требованиям:

1. принадлежать либо пространству имен тела цикла, либо объемлющему пространству имен, но так чтобы их значение не менялось на отрезке ξ;
2. среди них не должно быть обращений к анализируемому массиву;
3. если эта переменная принадлежит объемлющему пространству имен, ее значение не меняется на отрезке ξ, но эта переменная изменяется в цикле, то необходимо применить к ней механизм сохранения/восстановления значения, т.е. восстановить после цикла проверки.

В цикл-проверки добавляются только те операторы, которые обусловливают информационные зависимости между вхождениями анализируемого массива.

Если выражение-условие управляющей конструкции содержит обращение к анализируемому массиву, то преобразование отменяется.

Схема работы



Пример

Исходная программа

```
int A[100],B[10],L[10],R[10];
int x;

int main()
{
      int i;
      x=10;

      .
      .
      .
      .
      .


      for (i=1; i<=10; i=i+1)
      {
            A[R[i-1]]=A[R[i-1]]+x;
            B[i-1]=A[L[i-1]];
      }
      return i;
}
```

Преобразованная программа

```
int A[100];
int B[10];
int L[10];
int R[10];
int x;

int make_analysis( int tw, int n, int *A_w, int *A_r, int *A_np )
{
      int i;
      int tm = 0;
      int present_dep = 0;
      for ( i = 1; i < n; i = i + 1 )
          if ( A_w[i] ) tm = tm + 1;
      i = 1;
      while ( !present_dep && i < n );
            {
                  present_dep = present_dep || A_w[i] && A_r[i];
                  i = i + 1;
            }
      if ( present_dep ) return -1;
      else if ( tw == tm ) return 0;
      i = 1;
      present_dep = 0;
      while ( !present_dep && i < n );
            {
                  present_dep = present_dep || A_w[i] && A_np[i];
                  i = i + 1;
            }
    if ( present_dep ) return -2;
    else return 1;
}

int main()
{
      int i;
      int A_read_2[100];
      int A_write_3[100];
      int A_not_priv_4[100];
      int A_not_redux_5[100];
      int resultTesting_A_7;
      {
            int i_1;
            int tw_A_6;

            for (i_1=0; (i_1<99); (++i_1))
            {
                  A_read_2[i_1]=0;
                  A_write_3[i_1]=0;
                  A_not_priv_4[i_1]=0;
                  A_not_redux_5[i_1]=0;
            }

            for (i=1; (i<=10); i=(i+1))
            {
                  A_not_priv_4[R[(i-1)]]=1;
                  if ((A_write_3[L[(i-1)]]!=i))
                  {
                        A_read_2[L[(i-1)]]=i;
                        A_not_priv_4[L[(i-1)]]=1;
                  }
                  A_write_3[R[(i-1)]]=i;
```

```
            }

            for (i_1=0; (i_1<99); (++i_1))
                  if (A_write_3[i_1]) tw_A_6=(tw_A_6+1);

            //TODO: выполнить глобальную операцию редукции SUM для tw_A_6
            //TODO: выполнить слияние вспомогательных массивов
            resultTesting_A_7 = make_analysis_8(tw_A_6, 99, A_write_3,
                                                A_read_2, A_not_priv_4);
      }
      x=10;

      .
      .
      .
      .
      .
      .
      .
      .
      .

      if ((!resultTesting_A_7))
      {
            //отсутствуют все информационные зависимости
            for (i=1; (i<=10); i=(i+1))
            {
                  A[R[(i-1)]]=(A[R[(i-1)]]+x);
                  B[(i-1)]=A[L[(i-1)]];
            }
      }
      else
      {
            if ((resultTesting_A_7>0))
            {
                  //нет потоковых зависимостей
                  for (i=1; (i<=10); i=(i+1))
                  {
                        A[R[(i-1)]]=(A[R[(i-1)]]+x);
                        B[(i-1)]=A[L[(i-1)]];
                  }
            }
            else
            {
                  //присутствуют все виды информационных зависимостей
                  for (i=1; (i<=10); i=(i+1))
                  {
                        A[R[(i-1)]]=(A[R[(i-1)]]+x);
                        B[(i-1)]=A[L[(i-1)]];
                  }
            }
      }
      return i;
}
```