ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

PLC-2017

Тестирование преобразований программ в компиляторе с заданным критерием качества

Алымова Елена Владимировна

к.т.н., ст. преп. кафедры Алгебры и дискретной математики ИММиКН

Языки программирования и компиляторы
Всероссийская научная конференция памяти А.Л. Фуксмана
3–5 апреля 2017г., Ростов-на-Дону, Россия

Преобразование как объект тестирования

Преобразование – изменение фрагмента программы с целью улучшения её качества.

- Характеристики преобразования:
  - описание заменяемого фрагмента;
  - описание заменяющего фрагмента;
  - описание ограничений на информационные связи.
- Опасности:
  - нарушение синтаксиса результирующей программы;
  - нарушение семантики результирующей программы;
  - нарушение функциональной эквивалентности исходной и результирующей программ.

Цель работы:

- Исследование преобразования «*Вынос оператора из цикла*».
- Формализация условия применимости преобразования.
- Выбор критерия достаточности тестирования.
- Генерация набора тестов с заданным критерием достаточности.

Схема процесса тестирования эквивалентности преобразований программ

Полнота набора тестовых программ

- Исчерпывающее тестирование невозможно (в общем случае).
- Время на исполнение тестов ограничено.
- Необходим признак достаточности набора тестов.

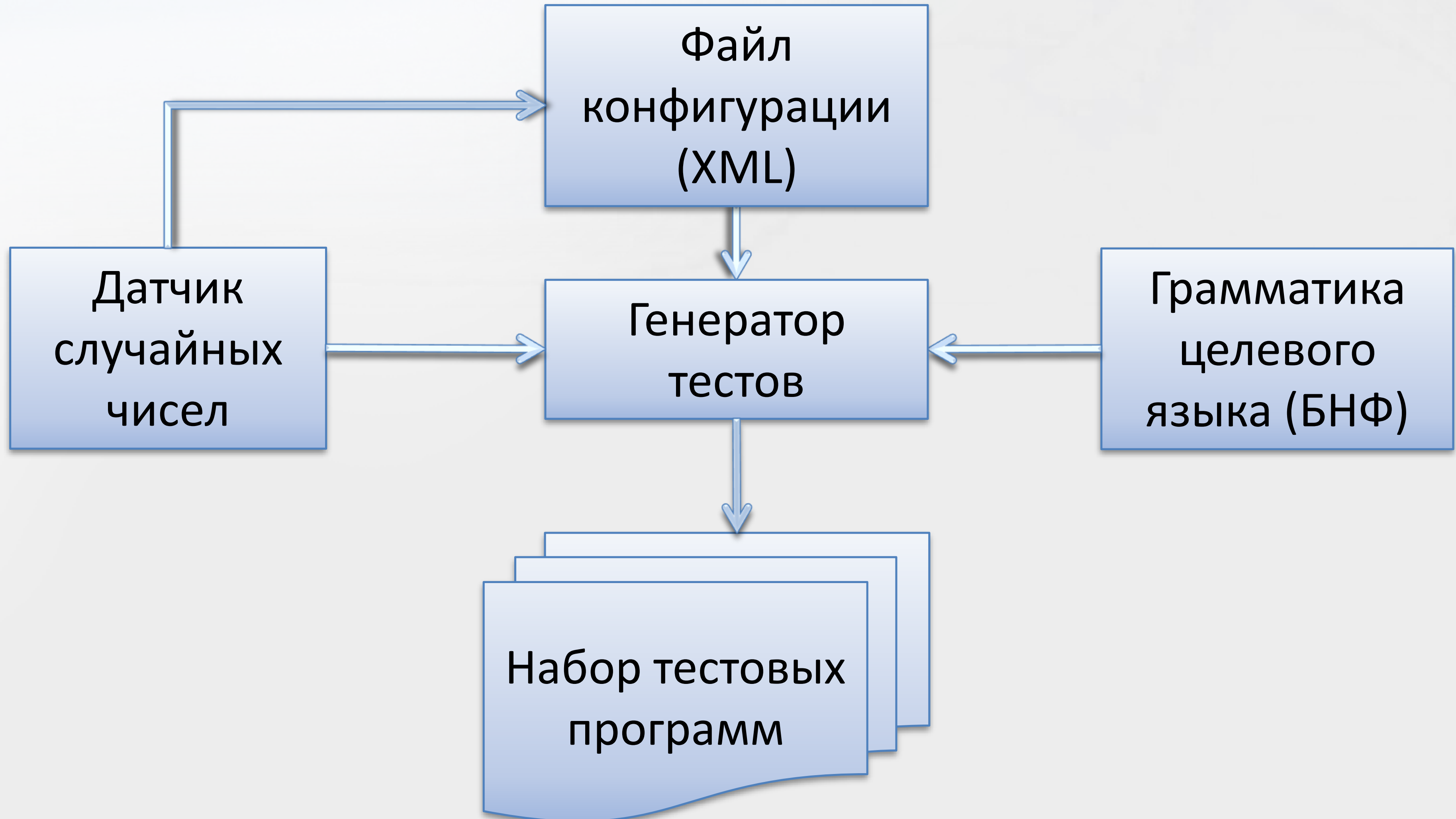
Критерий полноты набора тестов для преобразования

- Для любой комбинации из k операторов целевого языка существует тестовая программа в наборе, содержащая эту комбинацию.

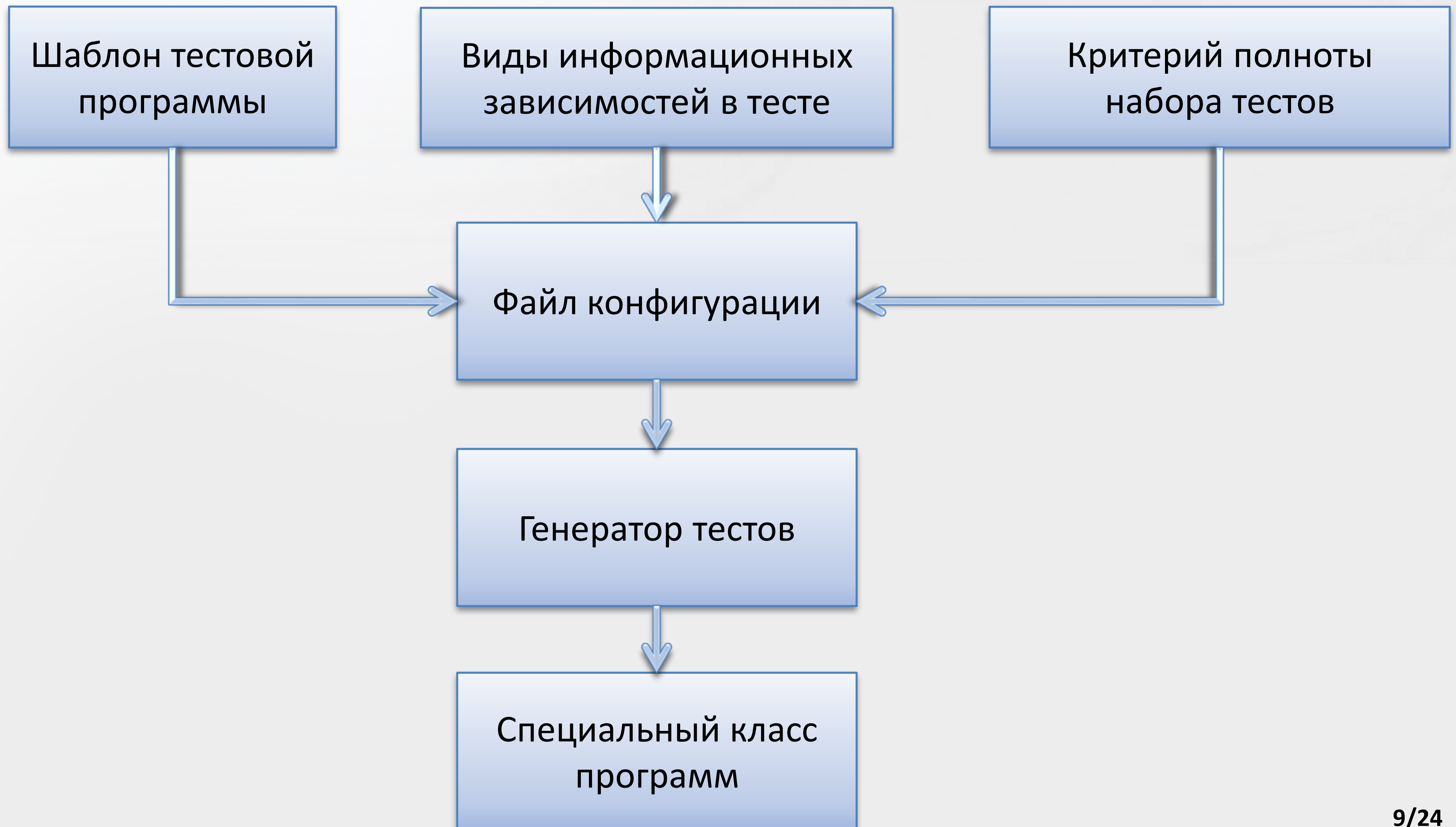
Тестируемые аспекты преобразования

- Распознавание фрагмента кода, к которому применимо преобразование.
- Ограничения на информационные зависимости в преобразуемом фрагменте кода.
- Семантическая корректность преобразованного кода.

Метод генерации тестов



Структура конфигурационного файла

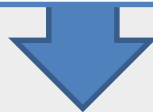


Преобразование «Вынос оператора из цикла»

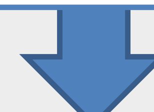
```
1  for (i = 0; i < N; i = i + 1)
2  {
3      FRAGMENT1(i);
4      FRAGMENT2(i);
5      FRAGMENT3(i);
6  }
```

В FRAGMENT2 все генераторы не зависят от счетчика цикла и в нём нет истинных циклически порожденных зависимостей.

Нет дуг графа информационных связей, ведущих **в** FRAGMENT2 **из** FRAGMENT1 и FRAGMENT3	Нет дуг графа информационных связей, ведущих **из** FRAGMENT2 **в** FRAGMENT1 и FRAGMENT3

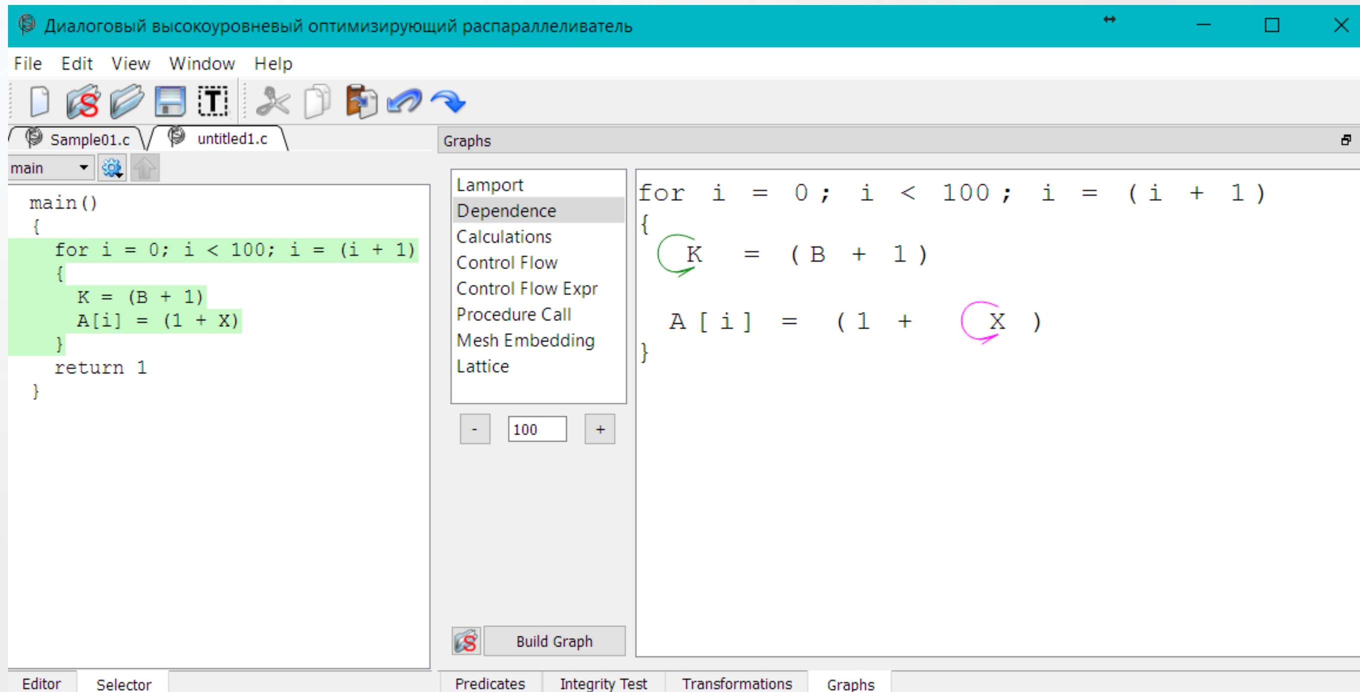


```
1  FRAGMENT2(N);
2  for (i = 0; i < N; i = i + 1)
3  {
4      FRAGMENT1(i);
5      FRAGMENT3(i);
6  }
```

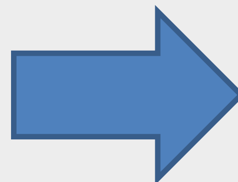


```
1  for (i = 0; i < N; i = i + 1)
2  {
3      FRAGMENT1(i);
4      FRAGMENT3(i);
5  }
6  FRAGMENT2(N);
```

Вынос оператора: идеальный случай



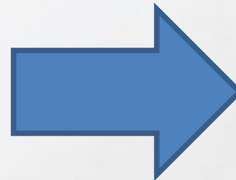

```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    for(i = 0; i < 100; i = i + 1)
6    {
7      K = B + 1;
8      A[i] = 1 + X;
9    }
10    return 1;
11 }
```



```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    K = B + 1;
6    for(i = 0; i < 100; i = i + 1)
7    {
8      A[i] = 1 + X;
9    }
10    return 1;
11 }
```

Вынос оператора: этапы идеального случая

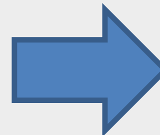
```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main()  {
5    for(i = 0; i < 100; i = i + 1)
6    {
7      K = B + 1;
8      A[i] = 1 + X;
9    }
10    return 1;
11 }
```



```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main()  {
5    K = B + 1;
6    for(i = 0; i < 100; i = i + 1)
7    {
8      A[i] = 1 + X;
9    }
10    return 1;
11 }
```

Разрезать цикл

```
1  int i, K, X;
2  int A[100];
3  int B[100];
4  int main()
5  {
6    for (i = 0; i < 100; i = i + 1)
7    {
8      K = B + 1;
9    }
10   for (i = 0; i < 100; i = i + 1)
11   {
12     A[i] = 1 + X;
13   }
14   return 1;
15 }
```

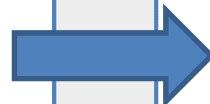


Удалить заголовок цикла

```
1  int i, K, X;
2  int A[100];
3  int B[100];
4  {
5    {
6      i = 99;
7      K = B + 1;
8    }
9    for (i = 0; i < 100; i = i + 1)
10   {
11     A[i] = 1 + X;
12   }
13   return 1;
14 }
```

Преобразование «Разрезание цикла»

```
int i;
int A[10];
int B[10];
int C[10];

int main()
{
     for (i=0; i<=9; i=i+1)
     {
        A[i]=B[i]+C[i+1];
        C[i]=A[i-1];
        B[i]=A[i-2]+1;
     }
}
```



```
int main()
{
   for (i=0; (i<=9); i=(i+1))
   {
      A[i]=(B[i]+C[(i+1)]);
   }
   for (i=0; (i<=9); i=(i+1))
   {
      C[i]=A[(i-1)];
   }
   for (i=0; (i<=9); i=(i+1))
   {
      B[i]=(A[(i-2)]+1);
   }
}
```

Граф информационных связей разрезания цикла

```
{
  i
  A
  B
  C
  main() : ST_INT
  {
    for i = 0; (i <= 9); i = (i + 1)
    {
      A[i] = (B[i] + C[(i + 1)])
      C[i] = A[(i – 1)]
      B[i] = (A[(i – 2)] + 1)
    }
  }
}
```

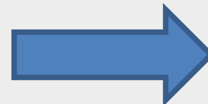
```
for( i = 0 ;  ( i <= 9 )  ;  i = ( i + 1 ) )
{
    A[i] = ( B[i] + C[(i+1)] ) ;

    C[i] = A[(i-1)] ;

    B[i] = ( A[(i-2)] + 1 ) ;
}
```

Тело цикла можно разрезать, если все дуги информационных связей между операторами в теле цикла направлены сверху-вниз.

Условия применимости преобразования «Разрезание циклов»

- Каждый из фрагментов программы S_1, …, S_k и S_{k+1},…, S_m имеют один вход и один выход.
- Не существует такой дуги графа информационных связей (v_1, v_2), что v_1 принадлежит S_i $(k + 1 \le i \le m)$, v_2 принадлежит S_j $(1 \le j \le k)$.

```
for (i = 1; i <= N; i = i + 1)
{
    S1
    ..........
    Sk
    S(k+1)
    ..........
    Sm
}
```



```
for (i = 1; i <= N; i = i + 1)
{
    S1
    ..........
    Sk
}
for (i = 1; i <= N, i = i + 1)
{
    S(k+1)
    ..........
    Sm
}
```

Вынос оператора: эквивалентность

Диалоговый высокоуровневый оптимизирующий распараллеливатель

File Edit View Window Help

Sample01.c untitled1.c | Graphs

main

```
main()
{
   for i = 0; i < 100; i = (i + 1)
   {
     X = (A[i] + X)
     K = (B[i] + X)
   }
   return 1
}
```

Lamport
Dependence
Calculations
Control Flow
Control Flow Expr
Procedure Call
Mesh Embedding
Lattice

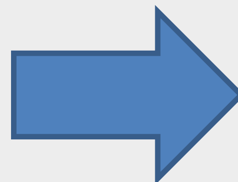
\- 100 +

Build Graph

```
for i = 0 ; i < 100 ; i = (i + 1)
{
X = (A [i] + X)

K = (B [i] + X)
}
```

Editor | Selector | Predicates | Integrity Test | Transformations | Graphs

```
1  int i, K, X;
2  int A[100];
3  int B [100];
4  int main() {
5     for(i = 0; i < 100; i = i + 1)
6     {
7        X = A[i] + X;
8        K = B[i] + X;
9     }
10 return 1;
11 }
```



```
1  int i, K, X;
2  int A[100];
3  int B [100];
4  int main() {
5     for(i = 0; i < 100; i = i + 1)
6     {
7        X = A[i] + X;
8     }
9        K = B[99] + X;
10 return 1;
11 }
```

Вынос оператора: эквивалентность. Почему?

Диалоговый высокоуровневый оптимизирующий распараллеливатель

File Edit View Window Help

untitled2_lloop.c | untitled1.c | Graphs

main

```
main()
{
  for i = 0; i < 100; i = (i + 1)
  {
    X = (A[i] + X)
    C[i] = X
    K = (B[i] + C[i])
  }
  return 1
}
```

Lamport
Dependence
Calculations
Control Flow
Control Flow Expr
Procedure Call
Mesh Embedding
Lattice

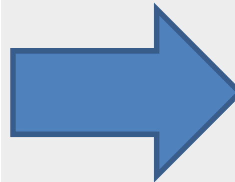
- 100 +

Build Graph

```
for i = 0 ; i < 100 ; i = ( i + 1 )
{
  X = ( A [ i ] + X )
  C [ i ] = X
  K = ( B [ i ] + C [ i ] )
}
```

Editor | Selector | Predicates | Integrity Test | Transformations | Graphs

```
1  int i, K, X;
2  int A[100]; int B[100]; int C[100];
3  int main()
4  {
5    for (i = 0; i < 100; i = i + 1)
6    {
7      X = A[i] + X;
8      C[i] = X;
9    }
10   for (i = 0; i < 100; i = i + 1)
11   {
12     K = B[i] + C[i];
13   }
14   return 1;
15 }
```



```
1  int i, K, X;
2  int A[100]; int B[100]; int C[100];
3  int main()
4  {
5    for (i = 0; i < 100; i = i + 1)
6    {
7      X = A[i] + X;
8      C[i] = X;
9    }
10   {
11     K = B[99] + C[99]; i = 99;
12   }
13   return 1;
14 }
```

Выноса оператора: эквивалентность

Диалоговый высокоуровневый оптимизирующий распараллеливатель

File Edit View Window Help

Sample01.c | untitled1.c

main

```
main()
{
  for i = 0; i < 100; i = (i + 1)
  {
    K = (L + X)
    X = (M * 4)
    A[i] = (B[i] + X)
  }
  return 1
}
```

Graphs

Lamport
Dependence
Calculations
Control Flow
Control Flow Expr
Procedure Call
Mesh Embedding
Lattice

- 100 +

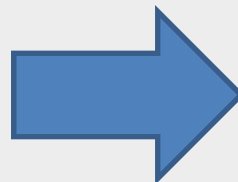
Build Graph

```
for i = 0 ; i < 100 ; i = ( i + 1 )
{
K = ( L + X )
X = ( M * 4 )
A [ i ] = ( B [ i ] + X )
}
```

Editor | Selector

Predicates | Integrity Test | Transformations | Graphs

```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    for(i = 0; i < 100; i = i + 1)
6    {
7      K = L + X;
8      X = M * 4;
9      A[i] = B[i] + X;
10   }
11   return 1;
12 }
```



```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    K = L + X;
6    X = M * 4;
7    for(i = 0; i < 100; i = i + 1)
8    {
9      A[i] = B[i] + X;
10   }
11   return 1;
12 }
```

Вынос оператора: эквивалентность. Почему?

Диалоговый высокоуровневый оптимизирующий распараллеливатель

File Edit View Window Help

untitled2_lloop.c | untitled1.c

main

```
main()
{
  __uni3[0] = X
  for i = 0; i < 100; i = (i + 1)
  {
    __uni3[(i + 1)] = (M * 4)
  }
  for i = 0; i < 100; i = (i + 1)
  {
    K = (L + __uni3[i])
  }
  for i = 0; i < 100; i = (i + 1)
  {
    A[i] = (B[i] + X)
  }
  X = __uni3[100]
  return 1
}
```

Graphs

Lamport
Dependence
Calculations
Control Flow
Control Flow Expr
Procedure Call
Mesh Embedding
Lattice

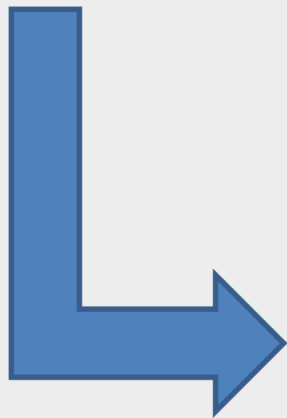
- 100 +

Build Graph

```
{
  __uni3 [ 0 ] = X
  for i = 0 ; i < 100 ; i = ( i + 1
  {
    __uni3 [ ( i + 1 ) ] = ( M *
  }
  for i = 0 ; i < 100 ; i = ( i + 1
  {
    K = ( L + __uni3 [ i ] )
  }
  for i = 0 ; i < 100 ; i = ( i + 1
  {
    A [ i ] = ( B [ i ] + X )
  }
  X = __uni3 [ 100 ]
  return 1
}
```

Editor | Selector

Predicates | Integrity Test | Transformations | Graphs



```
1
2  int i, K, X, L, M;
3  int A[100]; int B[100];
4  int main()
5  {
6    int __uni1[101];
7    __uni1[0] = X;
8    { i = 99; K = L + X; }
9    { i = 99; __uni1[(99 + 1)] = M * 4; }
10   for (i = 0; i < 100; i = i + 1)
11   {
12     A[i] = B[i] + __uni1[i];
13   }
14   X = __uni1[100];
15   return 1;
16 }
```

Вынос оператора: нарушение эквивалентности

Диалоговый высокоуровневый оптимизирующий распараллеливатель

File Edit View Window Help

Sample01.c | untitled1.c

main

```
main()
{
  for i = 0; i < 100; i = (i + 1)
  {
    X = (K + X)
    A[i] = X
  }
  return 1
}
```

Graphs

Lamport
Dependence
Calculations
Control Flow
Control Flow Expr
Procedure Call
Mesh Embedding
Lattice

- 100 +

Build Graph

```
for i = 0 ; i < 100 ; i = ( i + 1 )
{
  X = ( K + X )
  A [ i ] = X
}
```

Editor | Selector

Predicates | Integrity Test | Transformations | Graphs

```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    for(i = 0; i < 100; i = i + 1)
6    {
7      X = K + X;
8      A[i] = X;
9    }
10    return 1;
11 }
```

```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    X = K + X;
6    for(i = 0; i < 100; i = i + 1)
7    {
8      A[i] = X;
9    }
10    return 1;
11 }
```

Вынос оператора: нарушение эквивалентности

Диалоговый высокоуровневый оптимизирующий распараллеливатель

File Edit View Window Help

Sample01.c | untitled1.c

main

```
main()
{
  for i = 0; i < 100; i = (i + 1)
  {
    K = (B + X)
    X = (A[i] + X)
  }
  return 1
}
```

Graphs

Lamport
Dependence
Calculations
Control Flow
Control Flow Expr
Procedure Call
Mesh Embedding
Lattice

\- 100 +

Build Graph

```
for i = 0 ; i < 100 ; i = ( i + 1 )
{
  K = ( B + X )
  X = ( A [i ] + X )
}
```

Editor | Selector

Predicates | Integrity Test | Transformations | Graphs

```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    for(i = 0; i < 100; i = i + 1)
6    {
7      K = B + X;
8      X = A[i] + X;
9    }
10    return 1;
11 }
```

```
1  int i, K, L, M, X;
2  int A[100];
3  int B [100];
4  int main() {
5    K = B + X;
6    for(i = 0; i < 100; i = i + 1)
7    {
8      X = A[i] + X;
9    }
10    return 1;
11 }
```

Конфигурация для генерации набора тестов для «Выноса оператора из цикла»

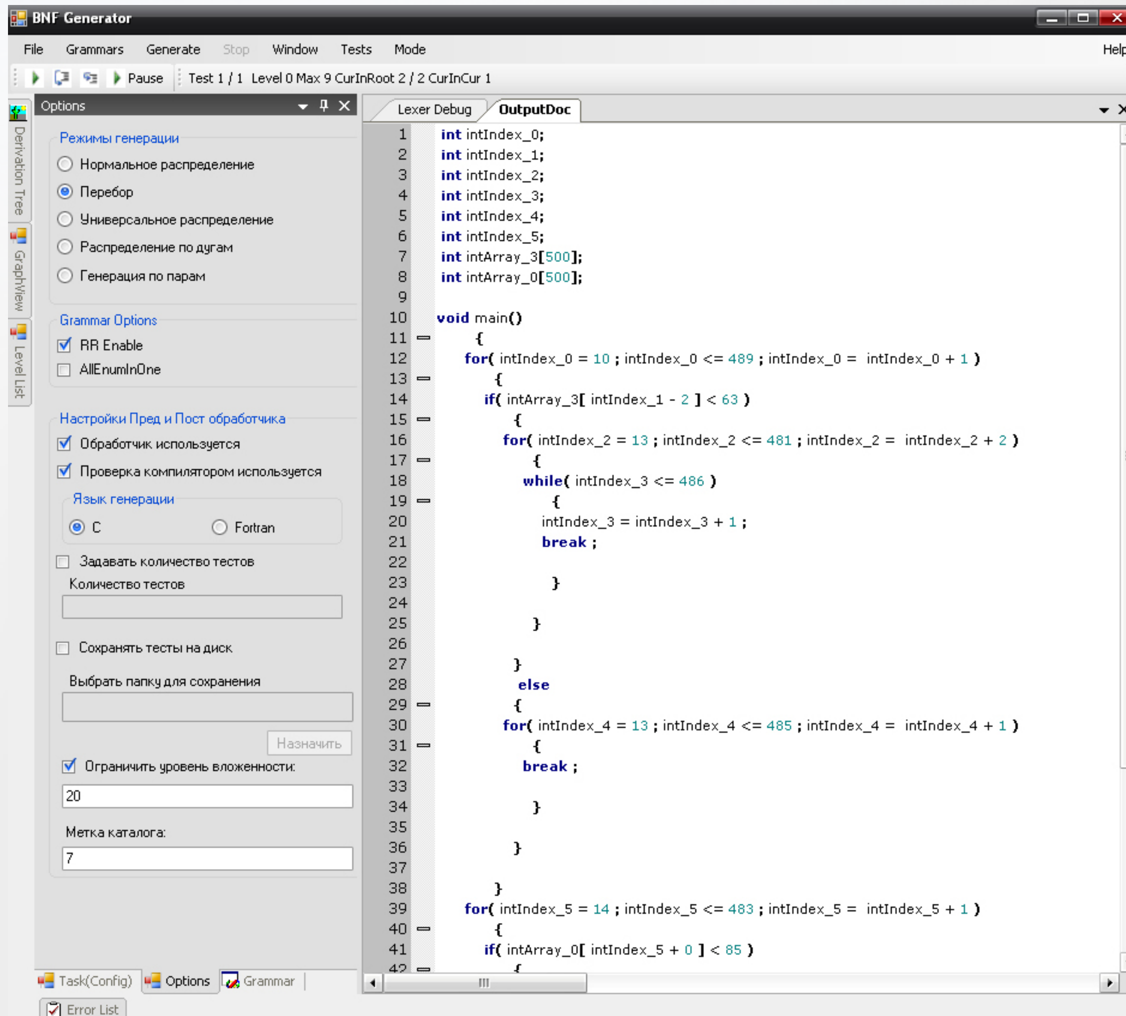
```xml
<?xml version="1.0" encoding="UTF-8"?>
<program class="op_cycle_remove">
    <params intArray="25" realArray="25" intIndex="30" />
    <body>
        <statFor m="4" statAssign="1" statIf="1" StatIfThenElse="1" StatWhile="1" StatSwitch="1" StatFor="1">
        <block>
                <dependency>
                <set type="in-in" direction="desc" cyclic="any" mode="any" />
                <set type="out-out" direction="desc"  cyclic="any" mode="any" />
                <set type="out-in" direction="desc" cyclic="any" mode="any" />
                <set type="in-out" direction="desc" cyclic="any" mode="any" />
                </dependency>
            <block>
                <dependency>
                <set type="in-in" direction="any" cyclic="no" mode="any" />
                <!-- Описание допустимых видов зависимостей -->
                </dependency>
                    <statAssign>
                        <gen mode="no-index" />
                        <use mode="any" />
                        <ops><![CDATA[* / % || && ^ << >>]]></ops>
                    </statAssign>
                    <!-- Описание допустимых операторов -->
            </block>
            <block>
                <dependency>
                <set type="in-in" direction="any" cyclic="any" mode="any" />
                <!-- Описание допустимых видов зависимостей -->
                </dependency>
                <!-- Описание допустимых операторов -->
            </block>
        </block>
        </statFor>
    </body>
</program>
```

Перебор четверок из шести операторов в теле цикла. Мощность набора тестов: $6^4 = 1296$

Пример тестовой программы для для «Выноса оператора из цикла»

```
1   #include <stdio.h>
2   /* Variable Declarations */
3   int main()
4   {
5     /* Initialization block */
6     for(intIndex_0 = 23; intIndex_0 <= 458;
7         intIndex_0 = intIndex_0 + 1)
8     {
9        realArray_3[intIndex_1] = intArray_2[intIndex_0] +
10       realArray_3[intIndex_1] + intArray_0[intIndex_1 + 5];
11       realArray_2[intIndex_3 + 1] =
12       intArray_12[intIndex_0 - 2] + 1.4 +
13       realArray_3[intIndex_1] * intArray_9[intIndex_1 + 2];
14     }
15     /* Result Output Block  */
16     return 1;
17  }
```

Генератор тестов: панель настроек



- Генерация цепочек по входной КС-грамматике.
- Различные режимы выбора альтернатив в правилах.
- Постобработка сгенерированного текста:
  - семантические ограничения целевого языка;
  - форматирование.
- Параметры сохранения набора тестов на диск.