На что способны современные статические анализаторы для C#

О Нас

- Чем занимается ИСП РАН?
  - Компиляторное технологии: анализ и оптимизация
  - R&D проекты для таких компаний, как Samsung, Intel

Валерий Игнатьев: к.ф.-м.н., 6 лет занимается статическим анализом.

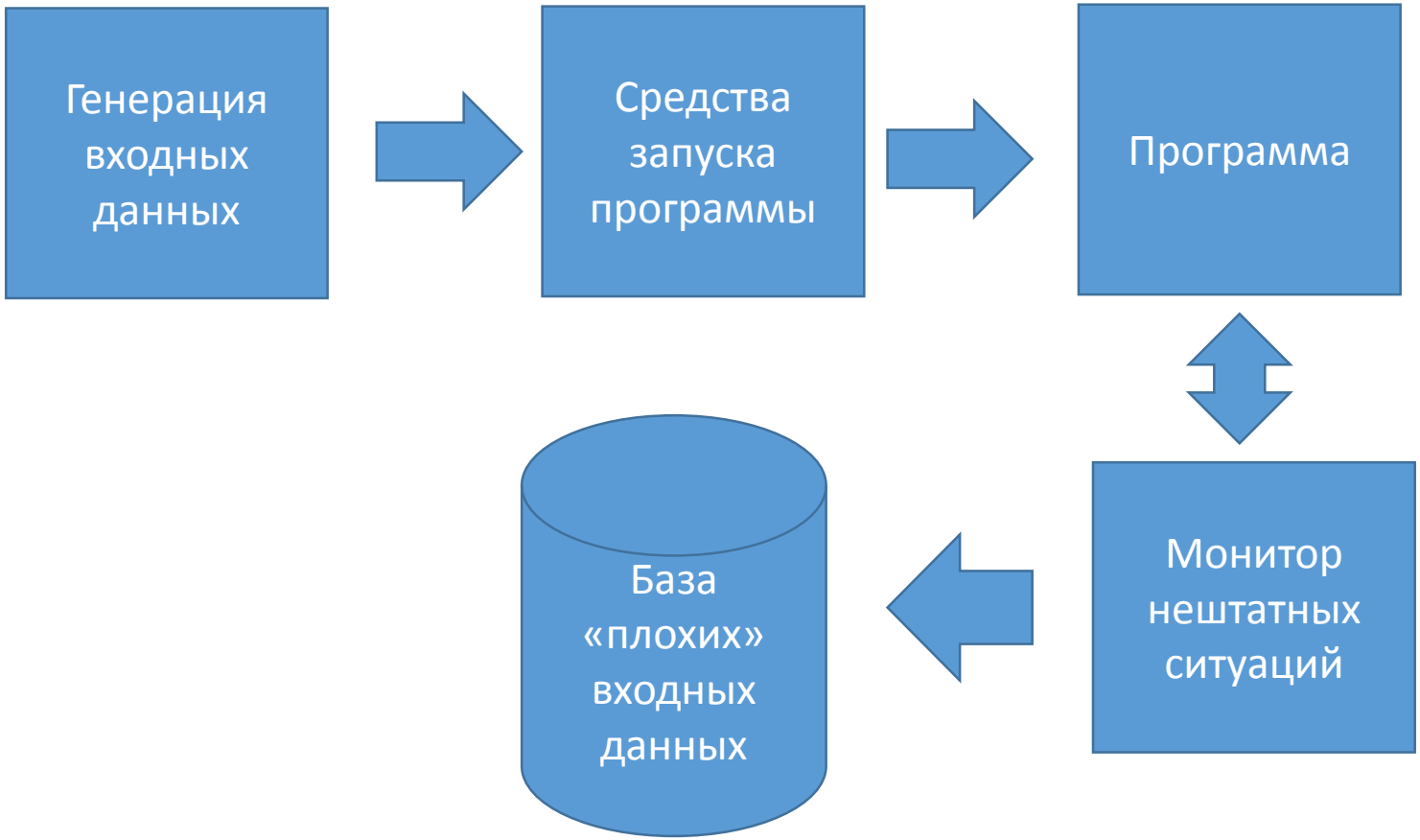
Владимир Кошелев: 4 года занимаюсь статическим анализом, пишу диссертацию про анализ C#.

Артем Борзилов: 2 года занимался динамическим анализом и деобфускацией, последние 2 года - статический анализ.

Автоматический анализ программ

- Задача: автоматический поиск ошибок в программах.
  - Поиск потенциальных ошибок (дефектов) в исходном коде программ
  - Поиск входных данных, на которых программа упадет (поиск уязвимостей)

Поиск уязвимостей: Фаззинг

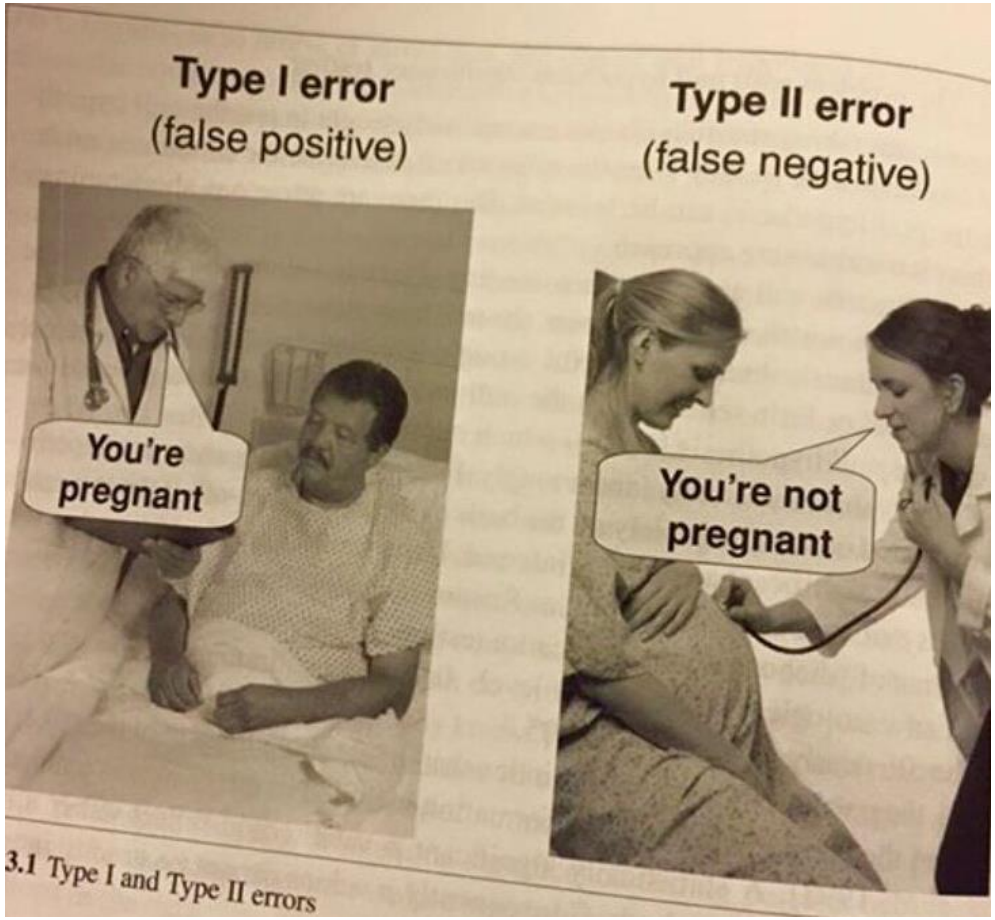


Разница между уязвимостью и дефектом

- Поиск уязвимостей - это прежде всего поиск «плохих» входных данных.

- Поиск дефектов - это поиск подозрительных мест в коде, входные данные искать не требуется.
  Поэтому его можно сделать менее точным и куда более быстрым, чем поиск уязвимостей.

Виды ошибок анализатора



3.1 Type I and Type II errors

Как оценивать качество анализа дефектов?

- Фиксируем время, отведенное для анализа, например, 10 минут для всего проекта.
- Фиксируем порог для ложных срабатываний, например, не более 50%.
- Тогда анализатор должен найти как можно больше срабатываний, соблюдая предыдущие два требования.

Методы поиска дефектов

- Анализ текста исходной программы
- Анализ внутреннего представления компилятора (например, абстрактных синтаксических деревьев)
- Анализ потенциальных путей выполнения программы

Поиск использований obsolete методов

```
class Foo
{
    [Obsolete]
    public void Request(string str)
    {
    // Some Code
    }
}
...
Foo foo;
...
foo.Request("abacaba")
```

Ищем .Request(

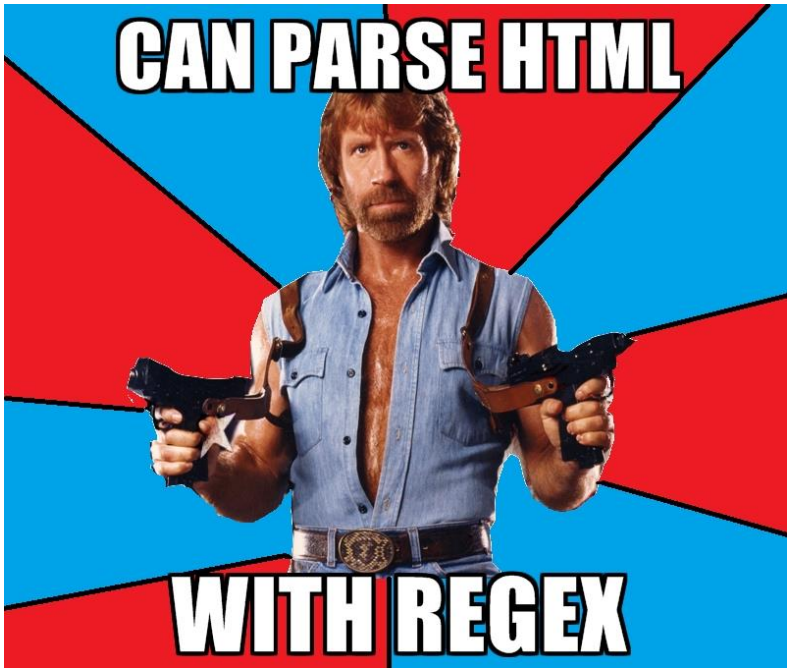
```
class Foo
{
    [Obsolete]
    public void Request(string str)
    {
    // Some Code
    }
}
...
Foo foo;
...
foo.Request("abacaba")
```

Проблема: поиск найдет использование всех методов Request.

```
class Bar {
    public Bar Request(string request, string out response)
    ...
}
class Baz {
   public void Request(Bar request)
   ...
}
...
Bar bar;
Baz baz;
baz.Request(bar.Request(s1, s2))
```

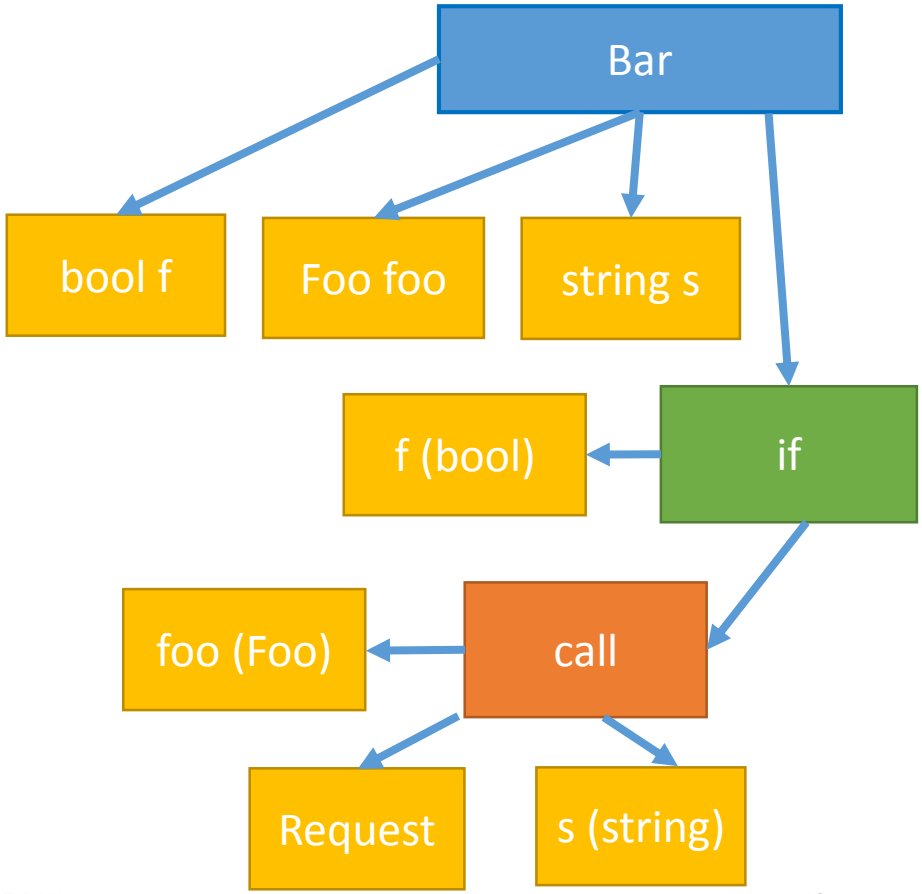
Можно, конечно, попытаться отфильтровать с помощью RegEx

Однако, RegEx не может распарсить даже HTML, куда там до C#.



Абстрактное синтаксическое дерево

```
static void Bar(bool f, Foo foo, string s)
{
    if (f)
    {
        foo.Request(s);
    }
}
```

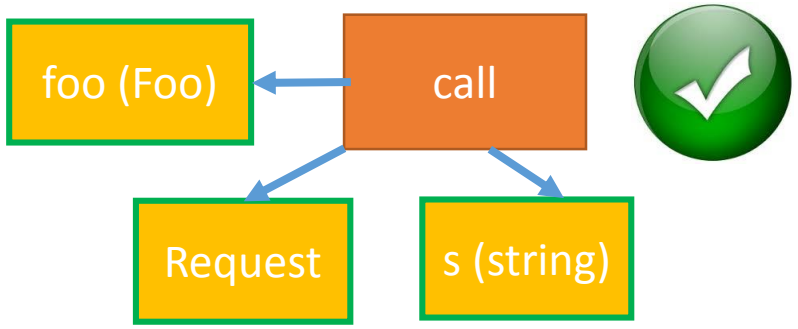


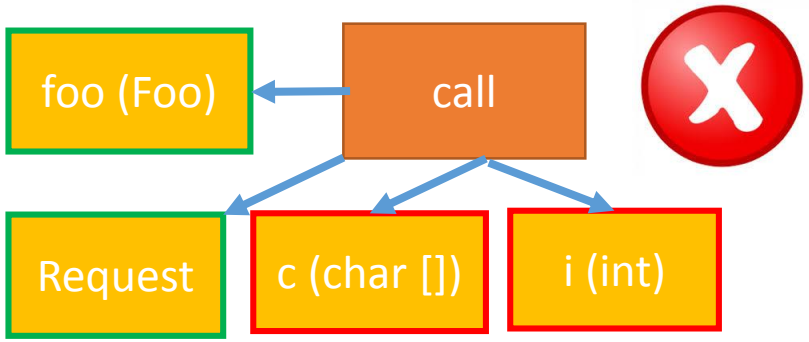
Метод | Оператор

Переменная | Выражение

Как найти все obsolete?

- Подписываемся на тип вершины call (для обхода AST используется паттерн Visitor).
- Проверяем, что тип переменной - Foo.
- Проверяем, что вызываемый метод – Request.





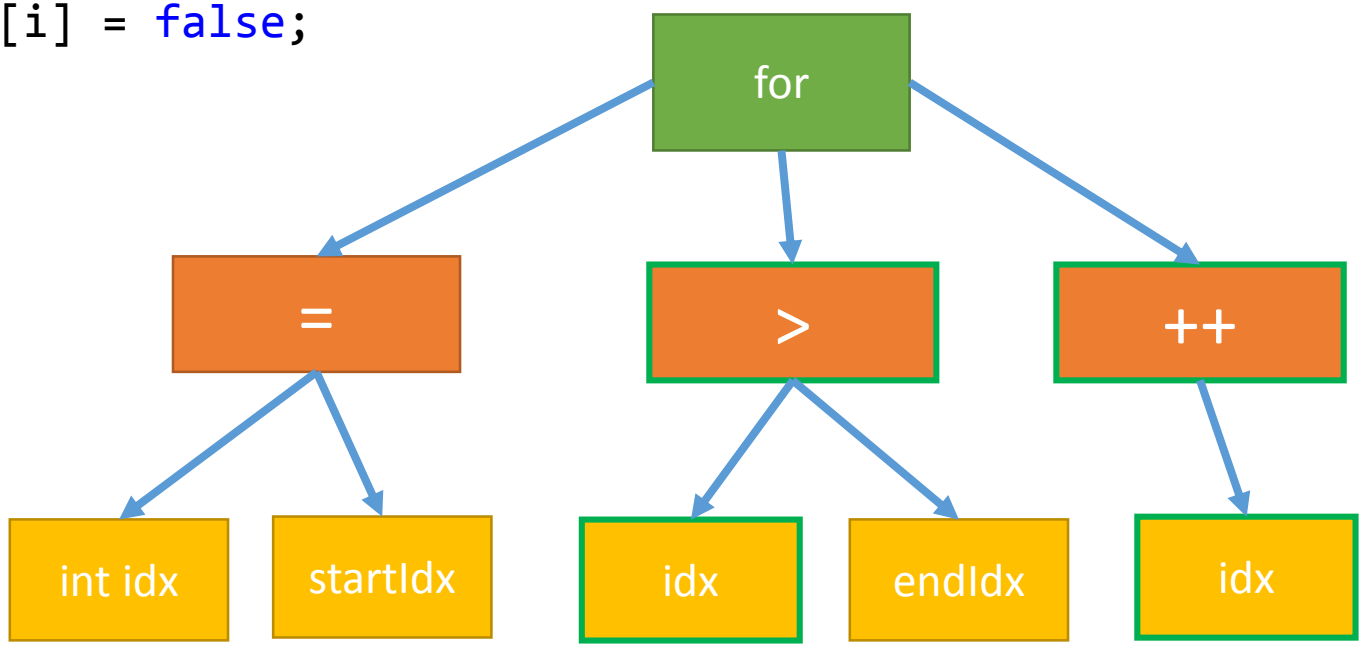
Поиск бесконечных циклов

Правило: Если индексная переменная имеет тип int и увеличивается в цикле, то в условии она должна проверяться на “меньше”.

```
for (int i = startIdx; i < endIdx; i++)
{
     expected[i] = false;
}
```

Пример ошибки

```
for (int idx = startIdx; idx > endIdx; idx++)
{
      expected[i] = false;
}
```



Можно ли найти с помощью AST обращение к null?

В простом случае – можно:

```
public static void AddRange<T>(this IList<T> list,
                               IEnumerable<T> values)
   {
       var lt = list as List<T>;
       if (lt != null)
           lt.AddRange(values);
       else  {
            foreach (var item in values)
            {
                lt.Add(item);
            }
        }
    }
```

А в реальности – есть проблемы

```
int numDiagnostics =
      diagnostics == null ? 0 : diagnostics.Count;
if (numDiagnostics > 0) {
    foreach (var diagEntry in diagnostics) {
            ...
    }
}
```

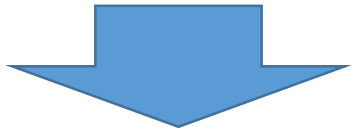
А в реальности – есть проблемы

```
int numDiagnostics =
      diagnostics == null ? 0 : diagnostics.Count;
if (numDiagnostics > 0) {
    foreach (var diagEntry in diagnostics) {
            ...
    }
}
```

Условие ошибки:

1) Если diagnostics == null, то numDiagnostics == 0

2) В foreach заходим, только если numDiagnostics > 0



Ошибка невозможна

Нужны методы анализа путей выполнения

И не только для поиска использований null!

Также можно искать:

- утечку ресурсов,
- использование освобожденных объектов (после Dispose),
- деление на ноль,
- ошибочное явное приведение типов,
- дефекты, делающие возможными SQL Injection, XML Injection и т.д.

Идея символьного выполнения

- Разрешим параметрам функции иметь неизвестные (символьные) значения:

```
1) int Sum(int a, int b)
2) {                    // a:= x_a, b:= x_b
3)     int c = a + b;
4)     int d = c*5;
5)     return d;
6) }
```

Идея символьного выполнения

- Разрешим параметрам функции иметь неизвестные (символьные) значения:

```
1) int Sum(int a, int b)
2) {                        // a:= x_a, b:= x_b
3)     int c = a + b; // a:= x_a, b:= x_b, c:= x_a+x_b
4)     int d = c*5;
5)     return d;
6) }
```

Идея символьного выполнения

- Разрешим параметрам функции иметь неизвестные (символьные) значения:

```
1) int Sum(int a, int b)
2) {                    // a:= x_a, b:= x_b
3)     int c = a + b;   // a:= x_a, b:= x_b, c:= x_a+x_b
4)     int d = c*5;     // a:= x_a, b:= x_b, c:= x_a+x_b, d:= (x_a+x_b)*5
5)     return d;
6) }
```

Идея символьного выполнения

- Разрешим параметрам функции иметь неизвестные (символьные) значения:

```
1) int Sum(int a, int b)
2) {                    // a:= x_a, b:= x_b
3)     int c = a + b;   // a:= x_a, b:= x_b, c:= x_a+x_b
4)     int d = c*5;     // a:= x_a, b:= x_b, c:= x_a+x_b, d:= (x_a+x_b)*5
5)     return d;        // a:= x_a, b:= x_b, c:= x_a+x_b, d:= (x_a+x_b)*5
6) }
```

Проблема: как обрабатывать if?

```
1) int Choose(int a)
2) {                                  // a:= x_a,
3)     int c = 0;
4)     if (a % 4 != 0)
5)         c = 1;
6)     else
7)         c = 2;
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Проблема: как обрабатывать if?

```
1) int Choose(int a)
2) {                                  // a:= x_a,
3)     int c = 0;                     // a:= x_a, c:= 0
4)     if (a % 4 != 0)
5)         c = 1;
6)     else
7)         c = 2;
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Проблема: как обрабатывать if?

```
1) int Choose(int a)
2) {                                  // a:= x_a,
3)     int c = 0;                     // a:= x_a, c:= 0
4)     if (a % 4 != 0)                // a:= x_a, c:= 0, куда идти?!
5)         c = 1;
6)     else
7)         c = 2;
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                              // a:= x_a,
3)     int c = 0;
4)     if (a % 4 != 0)
5)         c = 1;
6)     else
7)         c = 2;
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                          // a:= x_a,
3)     int c = 0;             // a:= x_a, c:= 0
4)     if (a % 4 != 0)
5)         c = 1;
6)     else
7)         c = 2;
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                          // a:= x_a,
3)     int c = 0;             // a:= x_a, c:= 0
4)     if (a % 4 != 0)        // a:= x_a, c:= 0
5)         c = 1;
6)     else
7)         c = 2;
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                          // a:= x_a,
3)     int c = 0;             // a:= x_a, c:= 0
4)     if (a % 4 != 0)        // a:= x_a, c:= 0
5)         c = 1;
6)     else
7)         c = 2;             // a:= x_a, c:= 2
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                          // a:= x_a,
3)     int c = 0;             // a:= x_a, c:= 0
4)     if (a % 4 != 0)        // a:= x_a, c:= 0
5)         c = 1;
6)     else
7)         c = 2;             // a:= x_a, c:= 2
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;             // a:= x_a, c:= 3
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                          // a:= x_a,
3)     int c = 0;             // a:= x_a, c:= 0
4)     if (a % 4 != 0)        // a:= x_a, c:= 0
5)         c = 1;
6)     else
7)         c = 2;             // a:= x_a, c:= 2
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;             // a:= x_a, c:= 3, не могли прийти!
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                          // a:= x_a,
3)     int c = 0;             // a:= x_a, c:= 0
4)     if (a % 4 != 0)        // a:= x_a, c:= 0, [x_a % 4 == 0]
5)         c = 1;
6)     else
7)         c = 2;
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Давайте определимся с порядком переходов

```
1) int Choose(int a)
2) {                          // a:= x_a,
3)     int c = 0;             // a:= x_a, c:= 0
4)     if (a % 4 != 0)        // a:= x_a, c:= 0, [x_a % 4 == 0]
5)         c = 1;
6)     else
7)         c = 2;             // a:= x_a, c:= 2, [x_a % 4 == 0]
8)     if (a % 2 != 0 && c == 2)
9)         c = 3;
10)     return c;
11) }
```

Символьное состояние состоит из:

Значений переменных, выраженных через символьные значения

a:= x_a, c:= 2, [x_a % 4 != 0], (7)

Условий пройденных переходов (Предикат пути)

a:= x_a, c:= 2, [x_a % 4 != 0], (7)

Текущей точки в программе

a:= x_a, c:= 2, [x_a % 4 != 0], (7)

Как с помощью этого искать использование null?

```
1) int Index(IList<string> diags, string elem)
2) {
3)     int nd; // diags:= x_d
4)     if (diags == null)
5)         nd = 0;
6)     else
7)         nd = diags.Count;
8)     if (nd >= 0)
9)     {
10)         return diags.IndexOf(elem);
11)    }
12)    return -1;
13) }
```

Как с помощью этого искать использование null?

```
1) int Index(IList<string> diags, string elem)
2) {
3)     int nd; // diags:= x_d
4)     if (diags == null) // diags:= x_d, [x_d = null]
5)         nd = 0;
6)     else
7)         nd = diags.Count;
8)     if (nd >= 0)
9)     {
10)         return diags.IndexOf(elem);
11)    }
12)    return -1;
13) }
```

Как с помощью этого искать использование null?

```
1) int Index(IList<string> diags, string elem)
2) {
3)     int nd; // diags:= x_d
4)     if (diags == null) // diags:= x_d, [x_d = null]
5)         nd = 0; // diags:= x_d, nd:= 0 [x_d = null]
6)     else
7)         nd = diags.Count;
8)     if (nd >= 0)
9)     {
10)         return diags.IndexOf(elem);
11)    }
12)    return -1;
13) }
```

Как с помощью этого искать использование null?

```
1) int Index(IList<string> diags, string elem)
2) {
3)     int nd; // diags:= x_d
4)     if (diags == null) // diags:= x_d, [x_d = null]
5)         nd = 0; // diags:= x_d, nd:= 0 [x_d = null]
6)     else
7)         nd = diags.Count;
8)     if (nd >= 0) // diags:= x_d, nd:= 0 [x_d = null, 0>= 0]
9)     {
10)         return diags.IndexOf(elem);
11)    }
12)    return -1;
13) }
```

Как с помощью этого искать использование null?

```
1) int Index(IList<string> diags, string elem)
2) {
3)     int nd; // diags:= x_d
4)     if (diags == null) // diags:= x_d, [x_d = null]
5)         nd = 0; // diags:= x_d, nd:= 0 [x_d = null]
6)     else
7)         nd = diags.Count;
8)     if (nd > 0) // diags:= x_d, nd:= 0 [x_d = null, 0 > 0]
9)     {
10)         return diags.IndexOf(elem);
11)    }
12)    return -1;
13) }
```

Как решить предикат пути?

- Для решения используются специальные сторонние программы: SMT-решатели (Z3, stp, etc.).
- Полученные формулы из символьных значений можно практически «как есть» отдавать решателям.
- Если есть решение, то решатель его явно приведет, если его нет, то скажет, что формула несовместна.

Как добиться масштабируемости анализа?

- Проблема: даже если не учитывать циклы, путей выполнения функции экспоненциально много.
- Если в функции 10 операторов If, то через них проходит 1024 различных пути.
- С учетом циклов, путей выполнения становится бесконечно много.

Можно объединять символьные состояния

```
if (p == null)
    c = 1; // p:= x_p, c:= 1, …, [… && x_p = null],
else
    c = 0; // p:= x_p, c:= 0, …, [… && x_p != null],

// p:= x_p, c:= (x_p=null ? 1: 0), [… && (x_p!=null || x_p=null),
```

Итого 1024 возможных пути превращаются в 10 объединений.

Анализ циклов

На практике, для получения хороших результатов достаточно анализировать лишь несколько итераций цикла.

Как следствие, работа с массивами и коллекциями поддерживается только частично.

Использование указателя после сравнения с null в цикле.

```
public int IndexOf(T item, IList<int> list)
{
      for (int i = fromIndex, fakeIndex = 0; i < toIndex; i++,
                fakeIndex++)
      {
           var current = list[i];
           if (current == null && item == null)
                return fakeIndex;
           if (current.Equals(item))
                return fakeIndex;
       }
       return -1;
}
```

Использование нулевого указателя: нужно поддерживать вызовы функции

```
if (fcontext == null) {
            // some code here
} else  {
            // some code there
}
vs.CreateWeight(fcontext, weightSearcher);

----------------------------------------

public override void CreateWeight(IDictionary context,
                                  IndexSearcher searcher)
{
    Weight w = searcher.CreateNormalizedWeight(q);
    context[this] = w;
}
```

Как организовать межпроцедурный анализ?

- **Стандартный подход:** при вызове функции начинать её символьное выполнение с текущим символьным состоянием.
  - Плюс: точно
  - Минус: очень медленно

- **Альтернативный подход:** заранее построить «резюме» для всех вызванных функций и использовать их вместо повторного анализа:
  - Плюс: достаточно быстро
  - Минус: неточно

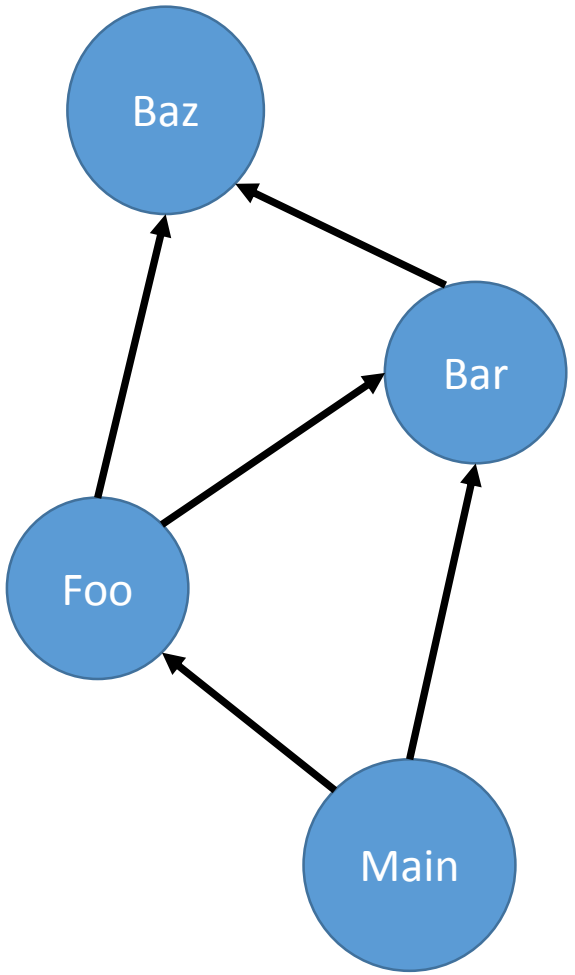
Какую информацию содержит резюме?

```
string Foo(IList<int> list, out string str, TextWriter wr)
```

- К объекту wr будет обращение, если он не null
- К объекту list будет обращение, если wr != null
- После выполнения вызова, str гарантированно не null.
- Функция может вернуть null
- Функция может бросить IOException

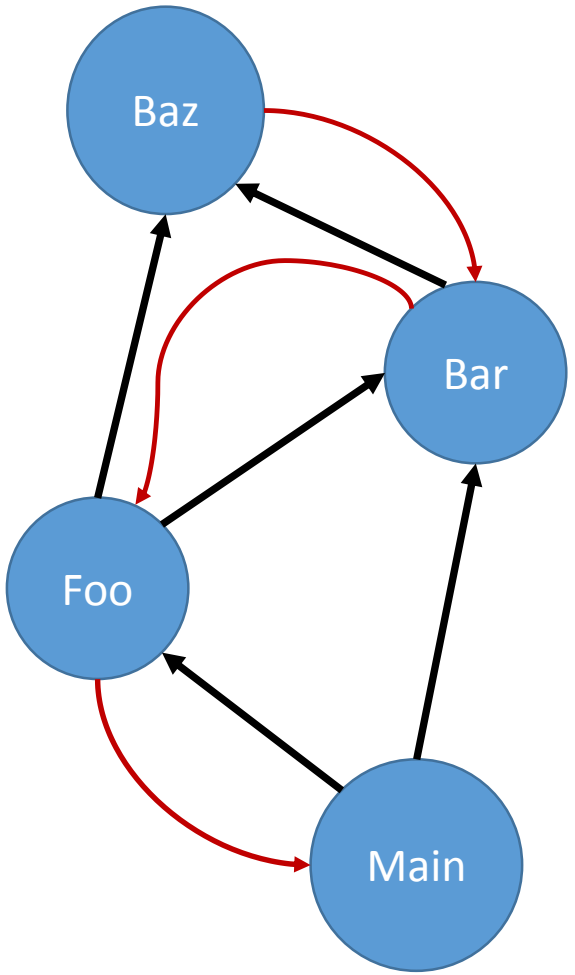
Что такое граф вызовов?

```
public void Baz() {

}

public void Main() {
    Foo();
    Bar();
}

public void Bar() {
    Baz();
}

public void Foo() {
     Bar();
     Baz();
}
```



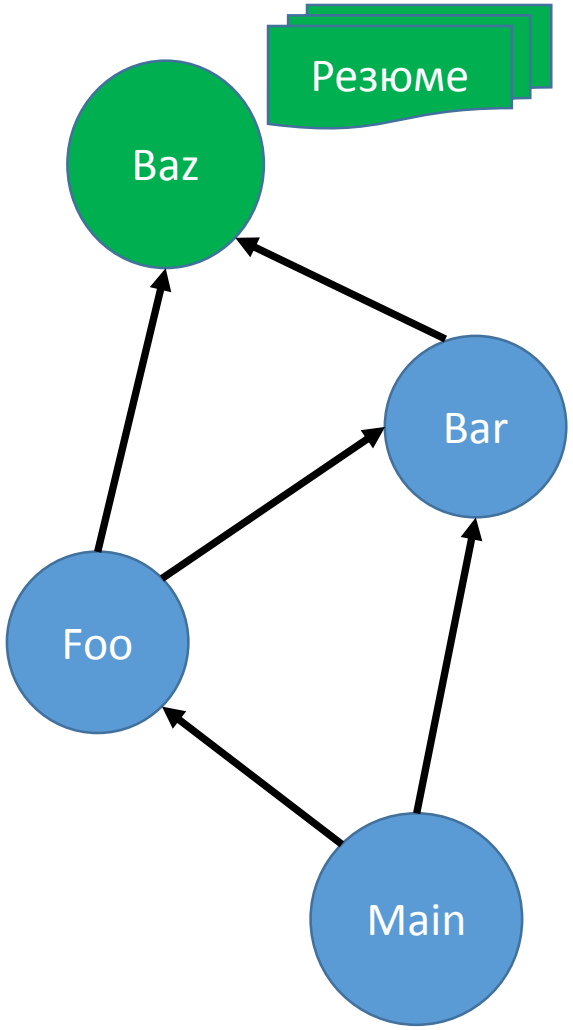
Обратный топологический порядок

```
public void Baz() {

}

public void Main() {
    Foo();
    Bar();
}

public void Bar() {
    Baz();
}

public void Foo() {
     Bar();
     Baz();
}
```



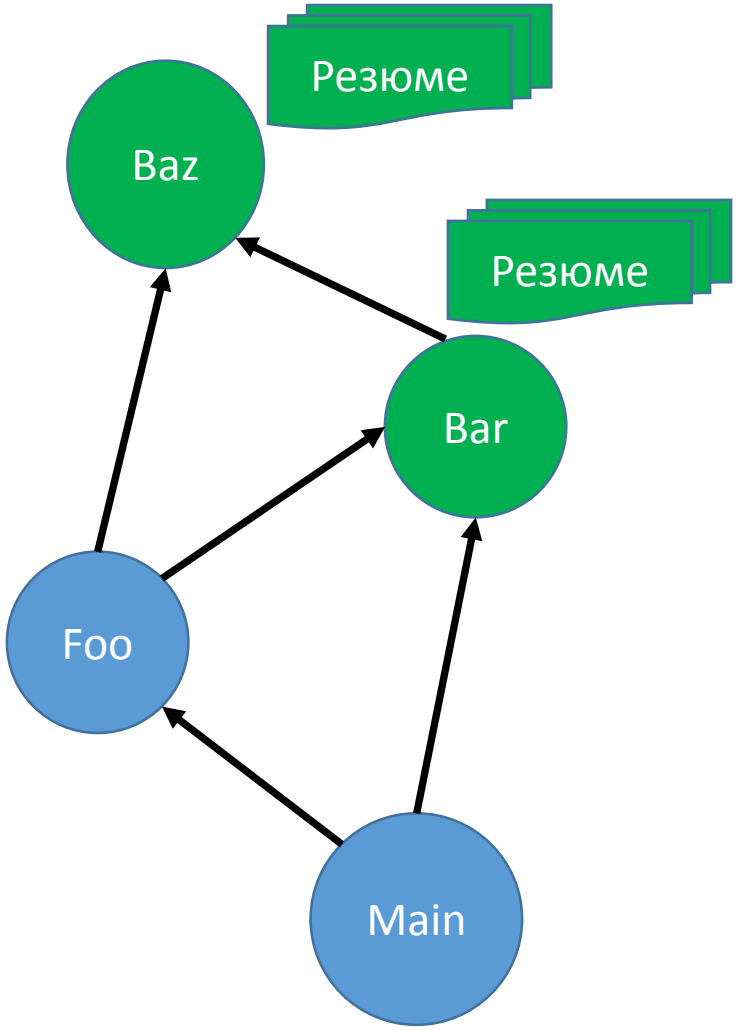
Порядок анализа

```
public void Baz() {

}

public void Main() {
    Foo();
    Bar();
}

public void Bar() {
    Baz();
}

public void Foo() {
     Bar();
     Baz();
}
```



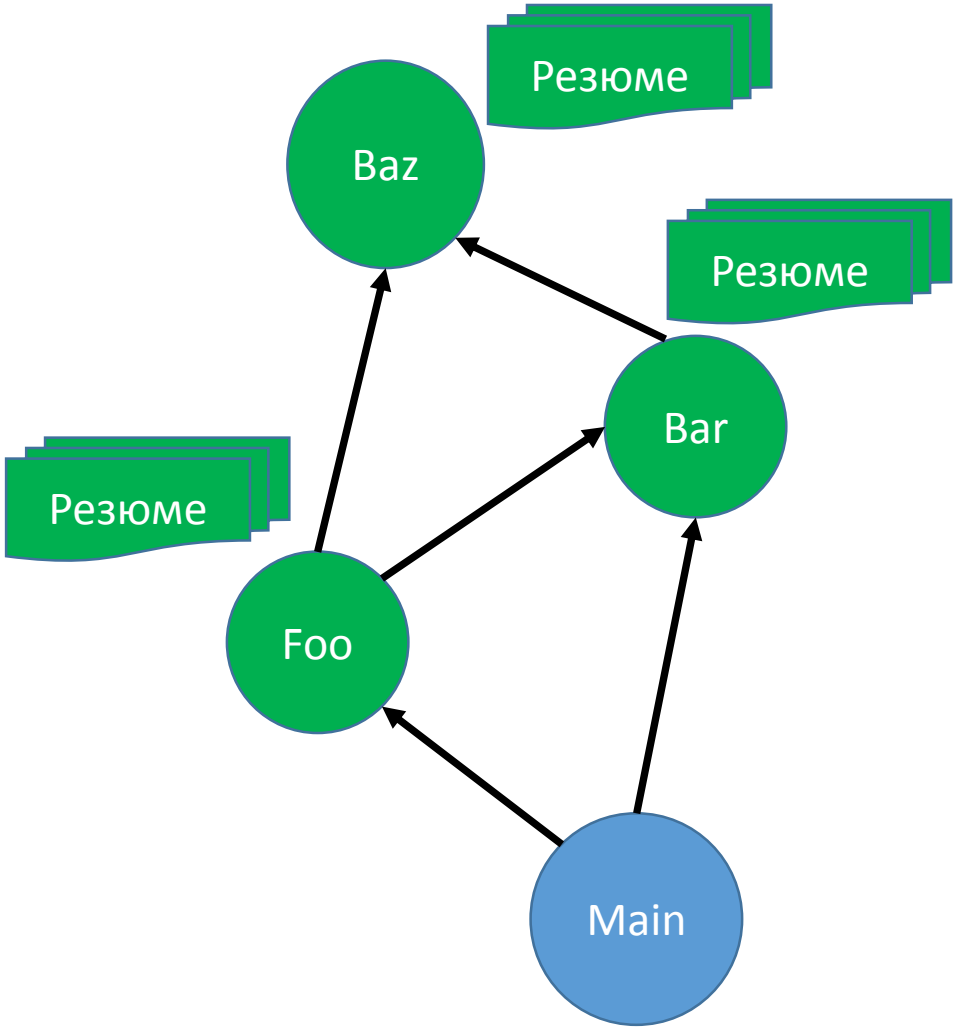
Порядок анализа

```
public void Baz() {

}

public void Main() {
    Foo();
    Bar();
}

public void Bar() {
    Baz();
}

public void Foo() {
     Bar();
     Baz();
}
```



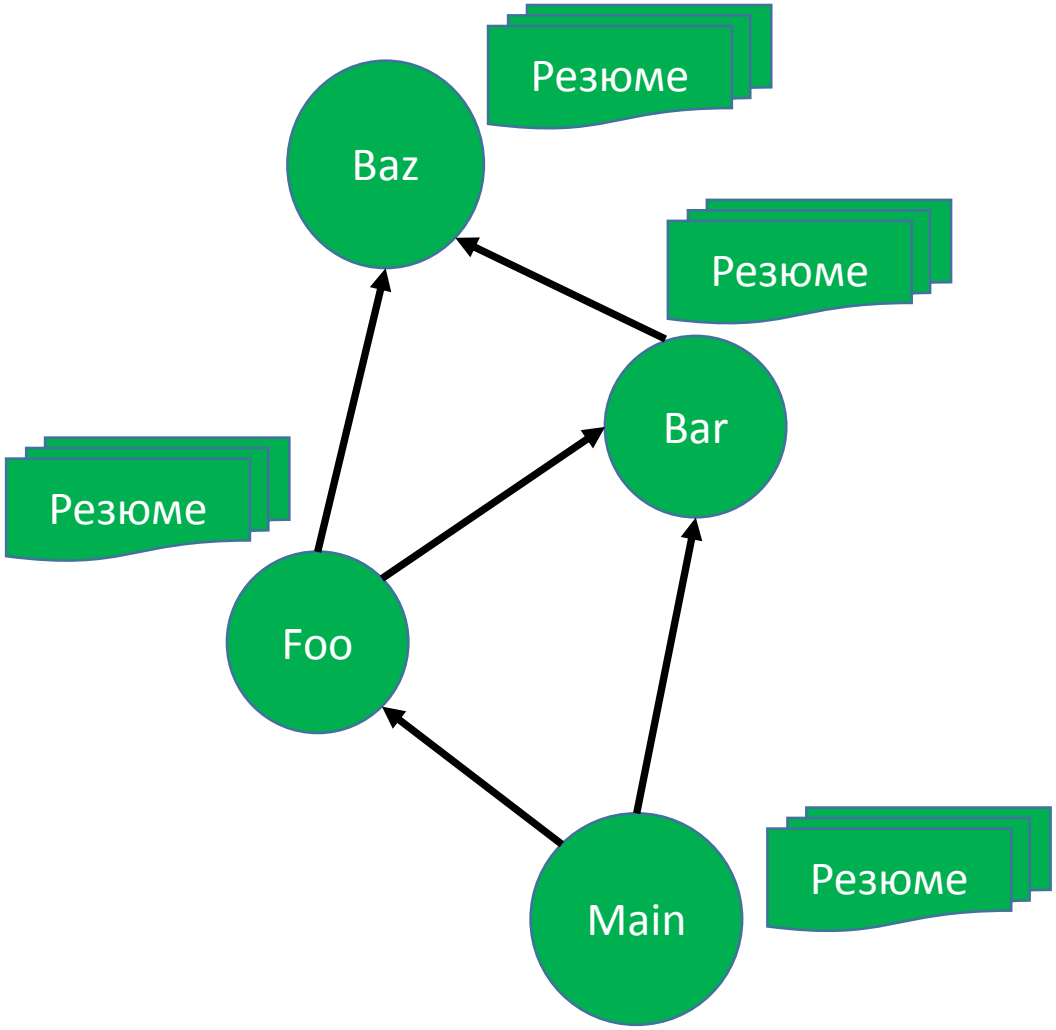
Порядок анализа

```
public void Baz() {

}

public void Main() {
    Foo();
    Bar();
}

public void Bar() {
    Baz();
}

public void Foo() {
     Bar();
     Baz();
}
```



Порядок анализа

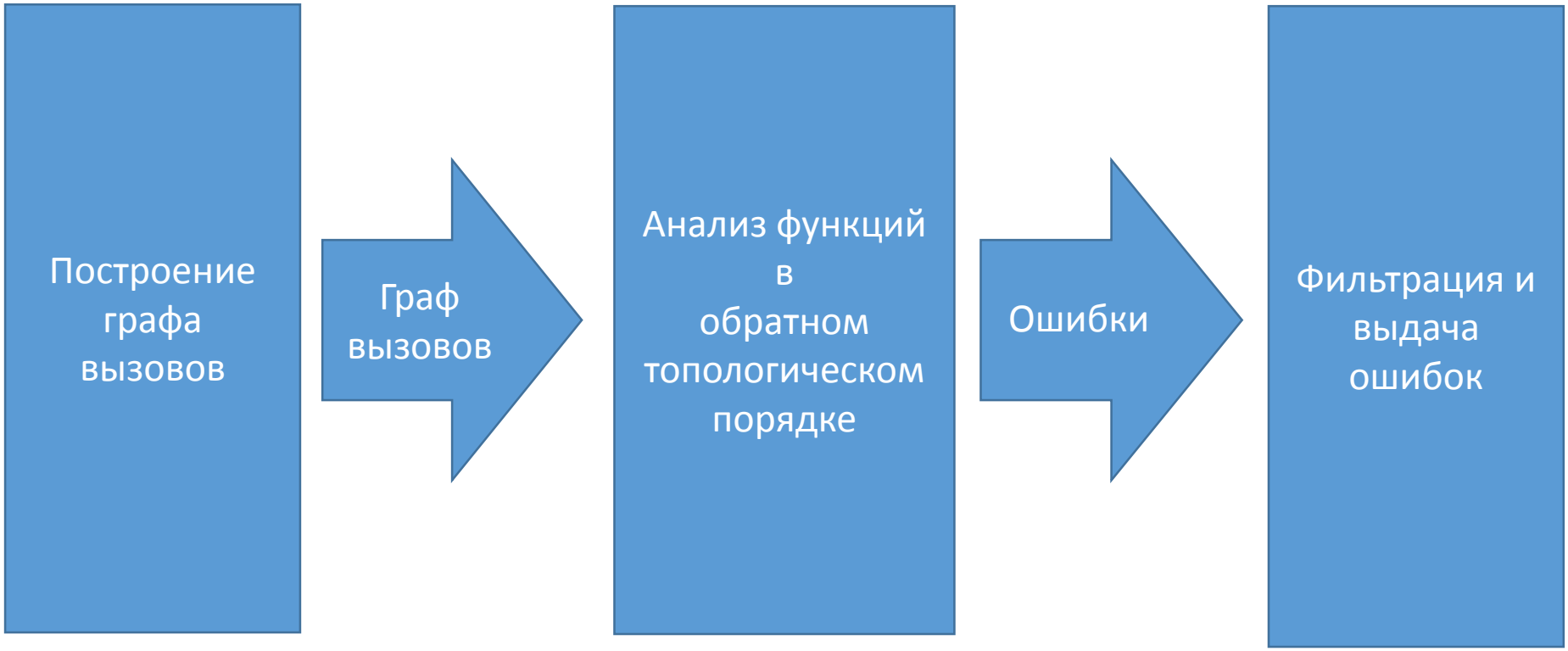
```
public void Baz() {

}

public void Main() {
    Foo();
    Bar();
}

public void Bar() {
    Baz();
}

public void Foo() {
     Bar();
     Baz();
}
```



Проблемы при построении графа вызовов

- Виртуальные вызовы
- Вызовы делегатов
- Вызовы внешних функций
- Рекурсия

Схема работы анализатора



Пример: Won’t fix

```
public string Foo(object obj)
{
    return (obj as Bar).Baz();
}
```

При анализе неизвестен контракт

```
public string Foo(IList<string> list) {
        int max = -1;
        string maxString = null;
        foreach (var elem in list) {
             if (elem.Length > max)  {
                  max = elem.Length;
                  maxString = elem;
             }
         }
         return maxString.ToUpper();
}
```

При анализе неизвестен контракт

```
public string Foo(IList<string> list) {
        int max = -1;
        string maxString = null;
        foreach (var elem in list) {
             if (elem.Length > max)  {
                  max = elem.Length;
                  maxString = elem;
             }
         }
         return maxString.ToUpper();
}
```

При анализе неизвестен контракт

```
public string Foo(IList<string> list) {
        Debug.Assert(list.Any())
        int max = -1;
        string maxString = null;
        foreach (var elem in list) {
             if (elem.Length > max)  {
                  max = elem.Length;
                  maxString = elem;
             }
         }
         return maxString.ToUpper();
}
```

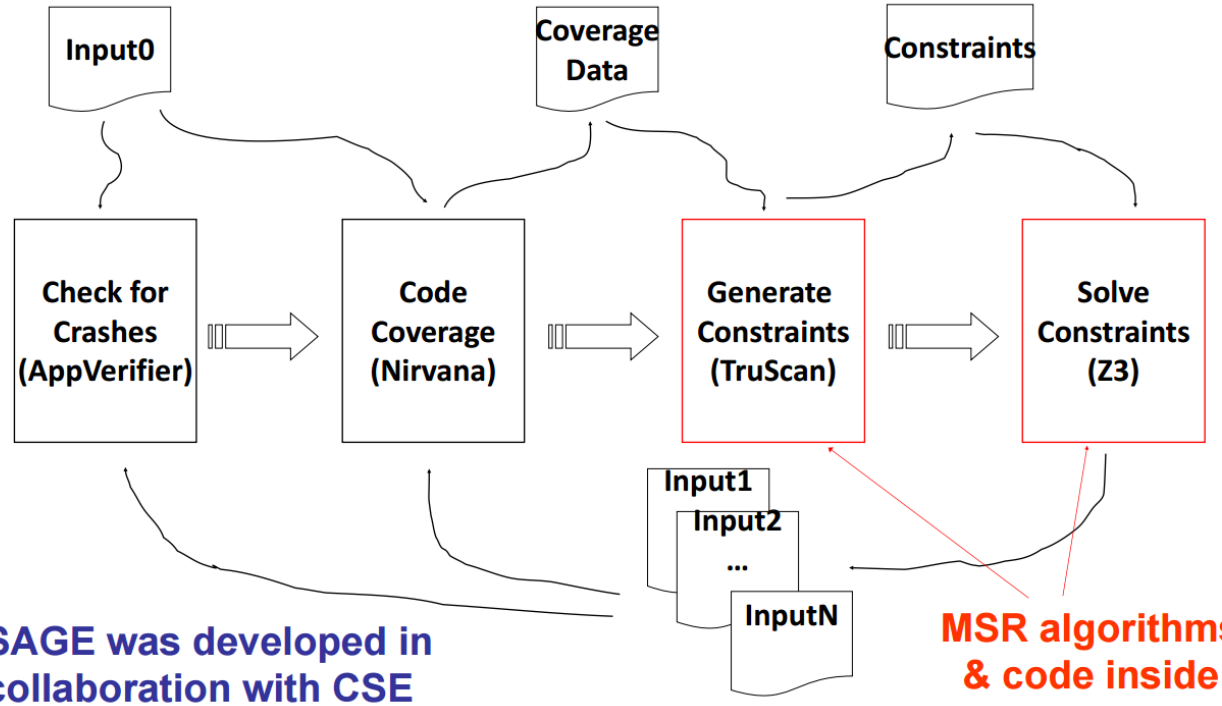
Причины ложных срабатываний

- Неточности в резюме функций;
- Наличие неизвестных вызовов;
- Наличие неявных контрактов функций;
- Наличие инвариантов классов.

Смешанное выполнение: Sage

Basic idea:

1. Run the program with first inputs,
2. gather constraints on inputs at conditional statements,
3. use a constraint solver to generate new test inputs,
4. repeat - possibly forever!



Смешанное выполнение: Sage

Impact: since 2007

- **200+ machine years (in largest fuzzing lab in the world)**
- **1 Billion+ constraints (largest SMT solver usage ever!)**
- **100s of apps, 100s of bugs (missed by everything else…)**
- **Ex: 1/3 of all Win7 WEX security bugs found by SAGE**
- **Bug fixes shipped quietly (no MSRCs) to 1 Billion+ PCs**
- **Millions of dollars saved (for Microsoft and the world)**
- **SAGE is now used daily in Windows, Office, etc.**

Спасибо за внимание!