Static code analysis tool for C#

Vladimir Koshelev

Senior Researcher

ISP RAS

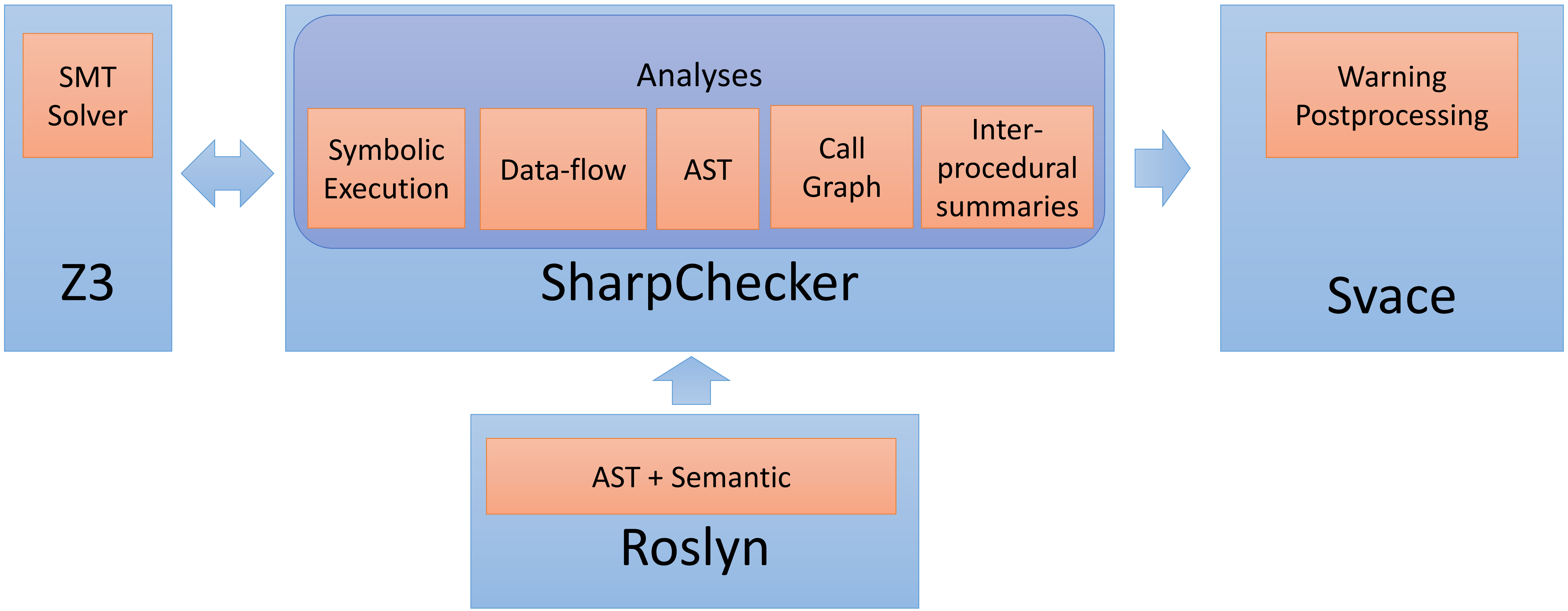
SharpChecker

- SharpChecker is production ready static code analysis tool for C#

- SharpChecker is based on the .NET Compiler Platform (“Roslyn”)

- SharpChecker performs complex interprocedural program analysis.
- It has good performance, for example, SharpChecker analyzes Roslyn 1.3.2 (prox. 2 millions lines of C# code) in 35 minutes, using intel i7 CPU.

SharpChecker architecture



AST Analyzers

- 38 different AST Analyzers are implemented in SharpChecker
- Examples:
  - Heuristic infinite loop detection (INFINITE_LOOP)
  - Using obsolete crypto algorithm (OBSOLETE_CRYPTO)
  - Copy paste errors (BAD_COPY_PASTE)

Bad Copy Paste Example (openBVE project)

```
for (int j = 0; j < Prototype.Mesh.Materials.Length; j++) {
        Result.Mesh.Materials[j] = Prototype.Mesh.Materials[j];
```

Start of original code block

```
        if (DaytimeTexture != null) {
                Result.Mesh.Materials[j].DaytimeTexture = DaytimeTexture;
        } else {
                Result.Mesh.Materials[j].DaytimeTexture = Prototype.Mesh.Materials[j].DaytimeTexture;
        }
```

In the expression DaytimeTexture != null variable DaytimeTexture possibly need to be replaced with NighttimeTexture after copy paste

Start of pasted copy

```
        if (DaytimeTexture != null) {
                Result.Mesh.Materials[j].NighttimeTexture = NighttimeTexture;
        } else {
                Result.Mesh.Materials[j].NighttimeTexture = Prototype.Mesh.Materials[j].NighttimeTexture;
        }
```

Bad Copy Paste Example (openBVE project)

```
for (int j = 0; j < Prototype.Mesh.Materials.Length; j++) {
        Result.Mesh.Materials[j] = Prototype.Mesh.Materials[j];
```

Start of original code block

```
        if (DaytimeTexture != null) {
                Result.Mesh.Materials[j].DaytimeTexture = DaytimeTexture;
        } else {
                Result.Mesh.Materials[j].DaytimeTexture = Prototype.Mesh.Materials[j].DaytimeTexture;
        }
```

In the expression DaytimeTexture != null variable DaytimeTexture possibly need to be replaced with NighttimeTexture after copy paste

Start of pasted copy

```
        if (DaytimeTexture != null) {
                Result.Mesh.Materials[j].NighttimeTexture = NighttimeTexture;
        } else {
                Result.Mesh.Materials[j].NighttimeTexture = Prototype.Mesh.Materials[j].NighttimeTexture;
        }
```

Data-flow analyzers

- Unused value analyzer:
  - Detects unused variable definitions
- Necessary condition analyzer (constant expression detection):
  - Calculates necessary condition for an entry point of each basic block
  - To express conditions we use predicate abstraction
  - To check that the expression is always constant we employ SMT provers.

Necessary condition example (WCell project)

```
public static string FormatTimeSecondsMinutes(int seconds)
{
        string time;
        if (seconds < 60)
        {
                time = seconds + " seconds";
        }
        else
        {
                var mins = seconds / 60;
                time = mins + (mins == 1 ? " minute" : " minutes");
                if (seconds % 60 != 0)
                {
Expression seconds == 1 is always false.
                        time += " and " + seconds + (seconds == 1 ? " second" : " seconds");
                }
        }
        return time;
}
```

Symbolic Execution

- We consider each method’s entry point as an entry point for symbolic execution.
- We analyze all paths that have at most one back edge.
- To reduce the complexity of path exploration, we merge symbolic states in basic blocks that have more than one predecessor.

Memory model

- We parameterize program state at entry point by set of symbolic variables.
- We use optimistic assumption that all references at entry point are not aliases.
- $\boldsymbol{S}$ is a set of symbolic variables. We describe memory as two maps:

$$\langle Local \mapsto \boldsymbol{S}, \boldsymbol{S} \times Field \mapsto \boldsymbol{S} \rangle$$

The first map symbolically models read/write operations with local variables.

The second map symbolically models read/write operations with the heap.

Conditions

- To achieve path sensitivity we model origin arithmetic operations by symbolic expressions
- For each basic block we calculate precise path condition
- For specified path and basic block, a path condition defines a set of concrete input states which leads the execution to the basic block following the path

Symbolic execution example

```
int Foo(Baz baz, int a)
{
    baz.X = a;
    if (baz.X + baz.Y > 0)
        return baz.X;
    else
        return baz.Y;
}
```

$\text{baz} \rightarrow \boldsymbol{baz}, \text{a} \rightarrow \boldsymbol{a}, \langle \boldsymbol{baz}, \text{X} \rangle \rightarrow \boldsymbol{x}, \langle \boldsymbol{baz}, \text{Y} \rangle \rightarrow \boldsymbol{y}$

Bold expressions are symbolic variables

Symbolic execution example

```
int Foo(Baz baz, int a)
{
    baz.X = a;
    if (baz.X + baz.Y > 0)
       return baz.X;
    else
       return baz.Y;
}
```

$\mathrm{baz} \to \boldsymbol{baz},\ \mathrm{a} \to \boldsymbol{a},\ \langle \boldsymbol{baz}, \mathrm{X} \rangle \to \boldsymbol{x}, \langle \boldsymbol{baz}, \mathrm{Y} \rangle \to \boldsymbol{y}$

$\mathrm{baz} \to \boldsymbol{baz},\ \mathrm{a} \to \boldsymbol{a},\ \langle \boldsymbol{baz}, \mathrm{X} \rangle \to \boldsymbol{a}, \langle \boldsymbol{baz}, \mathrm{Y} \rangle \to \boldsymbol{y}$

Bold expressions are symbolic variables

Symbolic execution example

```
int Foo(Baz baz, int a)
{
    baz.X = a;
    if (baz.X + baz.Y > 0)
        return baz.X;
    else
        return baz.Y;
}
```

Bold expressions are symbolic variables

$\text{baz} \to \boldsymbol{baz}, \text{a} \to \boldsymbol{a}, \langle \boldsymbol{baz}, \text{X} \rangle \to \boldsymbol{x}, \langle \boldsymbol{baz}, \text{Y} \rangle \to \boldsymbol{y}$

$\text{baz} \to \boldsymbol{baz}, \text{a} \to \boldsymbol{a}, \langle \boldsymbol{baz}, \text{X} \rangle \to \boldsymbol{a}, \langle \boldsymbol{baz}, \text{Y} \rangle \to \boldsymbol{y}$

Path Condition: $\boldsymbol{a + y > 0}$ $(\langle \boldsymbol{baz}, \text{X} \rangle \to \boldsymbol{a}, \langle \boldsymbol{baz}, \text{Y} \rangle \to \boldsymbol{y})$

Symbolic execution example

```
int Foo(Baz baz, int a)
{
    baz.X = a;
    if (baz.X + baz.Y > 0)
       return baz.X;
    else
       return baz.Y;
}
```

baz $\rightarrow$ $\boldsymbol{baz}$, a $\rightarrow$ $\boldsymbol{a}$, $\langle \boldsymbol{baz}, \mathrm{X}\rangle \rightarrow \boldsymbol{x}, \langle \boldsymbol{baz}, \mathrm{Y}\rangle \rightarrow \boldsymbol{y}$

baz $\rightarrow$ $\boldsymbol{baz}$, a $\rightarrow$ $\boldsymbol{a}$, $\langle \boldsymbol{baz}, \mathrm{X}\rangle \rightarrow \boldsymbol{a}, \langle \boldsymbol{baz}, \mathrm{Y}\rangle \rightarrow \boldsymbol{y}$

Path Condition: $\boldsymbol{a + y > 0}$ $(\langle \boldsymbol{baz}, \mathrm{X}\rangle \rightarrow \boldsymbol{a}, \langle \boldsymbol{baz}, \mathrm{Y}\rangle \rightarrow \boldsymbol{y})$

Path Condition: $\boldsymbol{a + y \leq 0}$ $(\langle \boldsymbol{baz}, \mathrm{X}\rangle \rightarrow \boldsymbol{a}, \langle \boldsymbol{baz}, \mathrm{Y}\rangle \rightarrow \boldsymbol{y})$

Bold expressions are symbolic variables

Null dereference examples

```
// NullReferenceException only when
// baz == null

public string Foo(Baz baz)
{
// Should not report
   return baz.ToString();
}
```

```
// always NullReferenceException


public string Foo(Baz baz)
{
   baz = null;
// Must report
   return baz.ToString();
}
```

Null dereference examples 2

```
// NullReferenceException only when
// f = true, f2 = true
public string Foo(Baz baz, bool f,
bool f2)
{
   if (f) {
      baz = null;
   }
   if (f2) // Should report?
      return baz.ToString();
   return null;
}
```

```
// baz can be null, however no
// path have null in baz and deref of
// baz.
public string Foo(Baz baz, bool f)
{
    if (f) {
        if (baz == null) Bar();
    } else {// Should report?
       return baz.ToString();
    }
    return null;
}
```

Error criteria for defect detection

- Common error criteria for null dereference:
- Let's assume that **symbolic** variable $\boldsymbol{x}$ is dereferenced in the basic block **B**.
- P_B is a path in a CFG that reaches basic block **B**
- $Cond_{P_B}(\bar{\boldsymbol{s}})$ – path condition

- Common error criteria for null dereference:

$$\exists P_B\left(\exists \bar{\boldsymbol{s}}\ Cond_{P_B}\ (\bar{\boldsymbol{s}})\right) \wedge \forall \bar{\boldsymbol{s}}(P_B(\bar{\boldsymbol{s}}) \rightarrow \boldsymbol{x} = \boldsymbol{null})$$

Symbolic execution checkers

- Null dereference
  - Explicit null dereference
  - Dereference after comparison
  - Reverse dereference
  - Interpocedural dereference

  …
- Resource Leak
- Invalid Cast
- Deadlock

Null dereference example (WCell project)

declared at

```
List<Exception> innerExceptions = null;
```

Step 1: Entering the foreach.

```
foreach (var inner in aggregateException.InnerExceptions)
{
    AggregateException innerAsAggregate = inner as AggregateException;
```

Step 2: Condition innerAsAggregate != null taking false branch

```
    if (innerAsAggregate != null)
    {
        AggregateException newChildAggregate = HandleRecursively(innerAsAggregate, predicate);
        if (newChildAggregate != null)
        {
            if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Also dereferenced in method call innerExceptions.Add()

```
            innerExceptions.Add(newChildAggregate);
        }
    }
```

Step 3: Condition !predicate(inner) taking true branch

```
    else if (!predicate(inner))
    {
```

Step 4: Condition innerExceptions != null taking false branch

```
        if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Value innerExceptions, which has null value, is dereferenced in method call innerExceptions.Add()

```
        innerExceptions.Add(inner);
```

Null dereference example (WCell project)

declared at

```
        List<Exception> innerExceptions = null;
```

Step 1: Entering the foreach.

```
        foreach (var inner in aggregateException.InnerExceptions)
        {
            AggregateException innerAsAggregate = inner as AggregateException;
```

Step 2: Condition innerAsAggregate != null taking false branch

```
            if (innerAsAggregate != null)
            {
                AggregateException newChildAggregate = HandleRecursively(innerAsAggregate, predicate);
                if (newChildAggregate != null)
                {
                    if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Also dereferenced in method call innerExceptions.Add()

```
                    innerExceptions.Add(newChildAggregate);
                }
            }
```

Step 3: Condition !predicate(inner) taking true branch

```
            else if (!predicate(inner))
            {
```

Step 4: Condition innerExceptions != null taking false branch

```
                if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Value innerExceptions, which has null value, is dereferenced in method call innerExceptions.Add()

```
                innerExceptions.Add(inner);
            }
```

Null dereference example (WCell project)

declared at

```
        List<Exception> innerExceptions = null;
```

Step 1: Entering the foreach.

```
        foreach (var inner in aggregateException.InnerExceptions)
        {
            AggregateException innerAsAggregate = inner as AggregateException;
```

Step 2: Condition innerAsAggregate != null taking false branch

```
            if (innerAsAggregate != null)
            {
                AggregateException newChildAggregate = HandleRecursively(innerAsAggregate, predicate);
                if (newChildAggregate != null)
                {
                    if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Also dereferenced in method call innerExceptions.Add()

```
                    innerExceptions.Add(newChildAggregate);
                }
            }
```

Step 3: Condition !predicate(inner) taking true branch

```
            else if (!predicate(inner))
            {
```

Step 4: Condition innerExceptions != null taking false branch

```
                if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Value innerExceptions, which has null value, is dereferenced in method call innerExceptions.Add()

```
                innerExceptions.Add(inner);
```

Null dereference example (WCell project)

declared at

```
        List<Exception> innerExceptions = null;
```

Step 1: Entering the foreach.

```
        foreach (var inner in aggregateException.InnerExceptions)
        {
            AggregateException innerAsAggregate = inner as AggregateException;
```

Step 2: Condition innerAsAggregate != null taking false branch

```
            if (innerAsAggregate != null)
            {
                AggregateException newChildAggregate = HandleRecursively(innerAsAggregate, predicate);
                if (newChildAggregate != null)
                {
                    if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Also dereferenced in method call innerExceptions.Add()

```
                    innerExceptions.Add(newChildAggregate);
                }
            }
```

Step 3: Condition !predicate(inner) taking true branch

```
            else if (!predicate(inner))
            {
```

Step 4: Condition innerExceptions != null taking false branch

```
                if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Value innerExceptions, which has null value, is dereferenced in method call innerExceptions.Add()

```
                innerExceptions.Add(inner);
```

Null dereference example (WCell project)

declared at

```
List<Exception> innerExceptions = null;
```

Step 1: Entering the foreach.

```
foreach (var inner in aggregateException.InnerExceptions)
{
    AggregateException innerAsAggregate = inner as AggregateException;
```

Step 2: Condition innerAsAggregate != null taking false branch

```
    if (innerAsAggregate != null)
    {
        AggregateException newChildAggregate = HandleRecursively(innerAsAggregate, predicate);
        if (newChildAggregate != null)
        {
            if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Also dereferenced in method call innerExceptions.Add()

```
            innerExceptions.Add(newChildAggregate);
        }
    }
```

Step 3: Condition !predicate(inner) taking true branch

```
    else if (!predicate(inner))
    {
```

Step 4: Condition innerExceptions != null taking false branch

```
        if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Value innerExceptions, which has null value, is dereferenced in method call innerExceptions.Add()

```
        innerExceptions.Add(inner);
    }
```

Null dereference example (WCell project)

declared at

```
List<Exception> innerExceptions = null;
```

Step 1: Entering the foreach.

```
foreach (var inner in aggregateException.InnerExceptions)
{
    AggregateException innerAsAggregate = inner as AggregateException;
```

Step 2: Condition innerAsAggregate != null taking false branch

```
    if (innerAsAggregate != null)
    {
        AggregateException newChildAggregate = HandleRecursively(innerAsAggregate, predicate);
        if (newChildAggregate != null)
        {
            if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Also dereferenced in method call innerExceptions.Add()

```
            innerExceptions.Add(newChildAggregate);
        }
    }
```

Step 3: Condition !predicate(inner) taking true branch

```
    else if (!predicate(inner))
    {
```

Step 4: Condition innerExceptions != null taking false branch

```
        if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Value innerExceptions, which has null value, is dereferenced in method call innerExceptions.Add()

```
        innerExceptions.Add(inner);
```

Null dereference example (WCell project)

declared at

```
List<Exception> innerExceptions = null;
```

Step 1: Entering the foreach.

```
foreach (var inner in aggregateException.InnerExceptions)
{
    AggregateException innerAsAggregate = inner as AggregateException;
```

Step 2: Condition innerAsAggregate != null taking false branch

```
    if (innerAsAggregate != null)
    {
        AggregateException newChildAggregate = HandleRecursively(innerAsAggregate, predicate);
        if (newChildAggregate != null)
        {
            if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Also dereferenced in method call innerExceptions.Add()

```
            innerExceptions.Add(newChildAggregate);
        }
    }
```

Step 3: Condition !predicate(inner) taking true branch

```
    else if (!predicate(inner))
    {
```

Step 4: Condition innerExceptions != null taking false branch

```
        if (innerExceptions != null) innerExceptions = new List<Exception>();
```

Value innerExceptions, which has null value, is dereferenced in method call innerExceptions.Add()

```
        innerExceptions.Add(inner);
```

Questions?

http://sharpchecker.ispras.ru

Email: sharpchecker@ispras.ru