Lomonosov
Moscow State University

Ivannikov
Institute for System Programming
of the RAS

Modeling of library functions in an industrial static code analyzer

Mikhail Belyaev, Egor Romanenkov, Valery Ignatyev

May 28, 2020

Introduction

Library modeling is important
for the precision of **interprocedural** static analysis

Related work

- JetBrains annotations
- Coverity code models
- Language-integrated features
- StubDroid summaries

Comparison criteria

- Scalability
- Performance influence
- Completeness
- Maintainability
- Ability to represent unique properties
- Possibility of manual correction
- Size
- Possibility of private code modeling

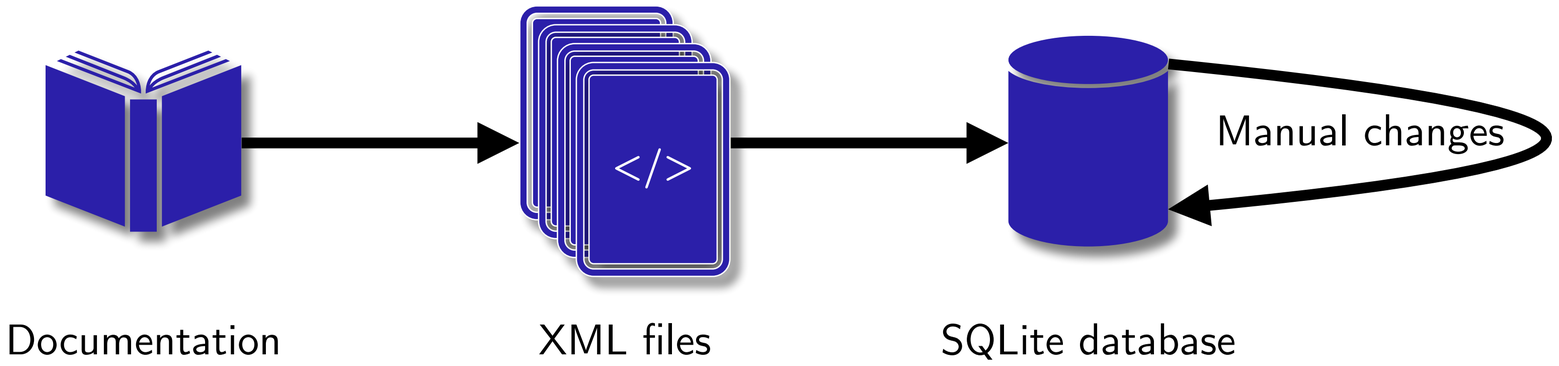
Approaches to library modeling

- Hardcoded semantics
- Property database
- Code models
- Library pre-analysis

Hardcoded semantics

- Comparison with null:
  - `string.IsNullOrEmpty(string)`
  - `string.IsNullOrWhitespace(string)`
  - `System.Nullable<T>.HasValue`
- Asserts
- Object comparison:
  - `object.Equals(object)`
  - `object.Equals(object, object)`
  - `object.ReferenceEquals(object, object)`
  - `IEqualityComparer.Equals(object, object)`
  - `IEquatable<T>.Equals(T)`
- `System.Environment.Exit(int)`
- `System.Nullable<T>.Value`

Property database



Can return null

Types of thrown exceptions

Parameters that must be non-null

Disposes `this`

Changes inner state

Uses multithreading

...

Code models

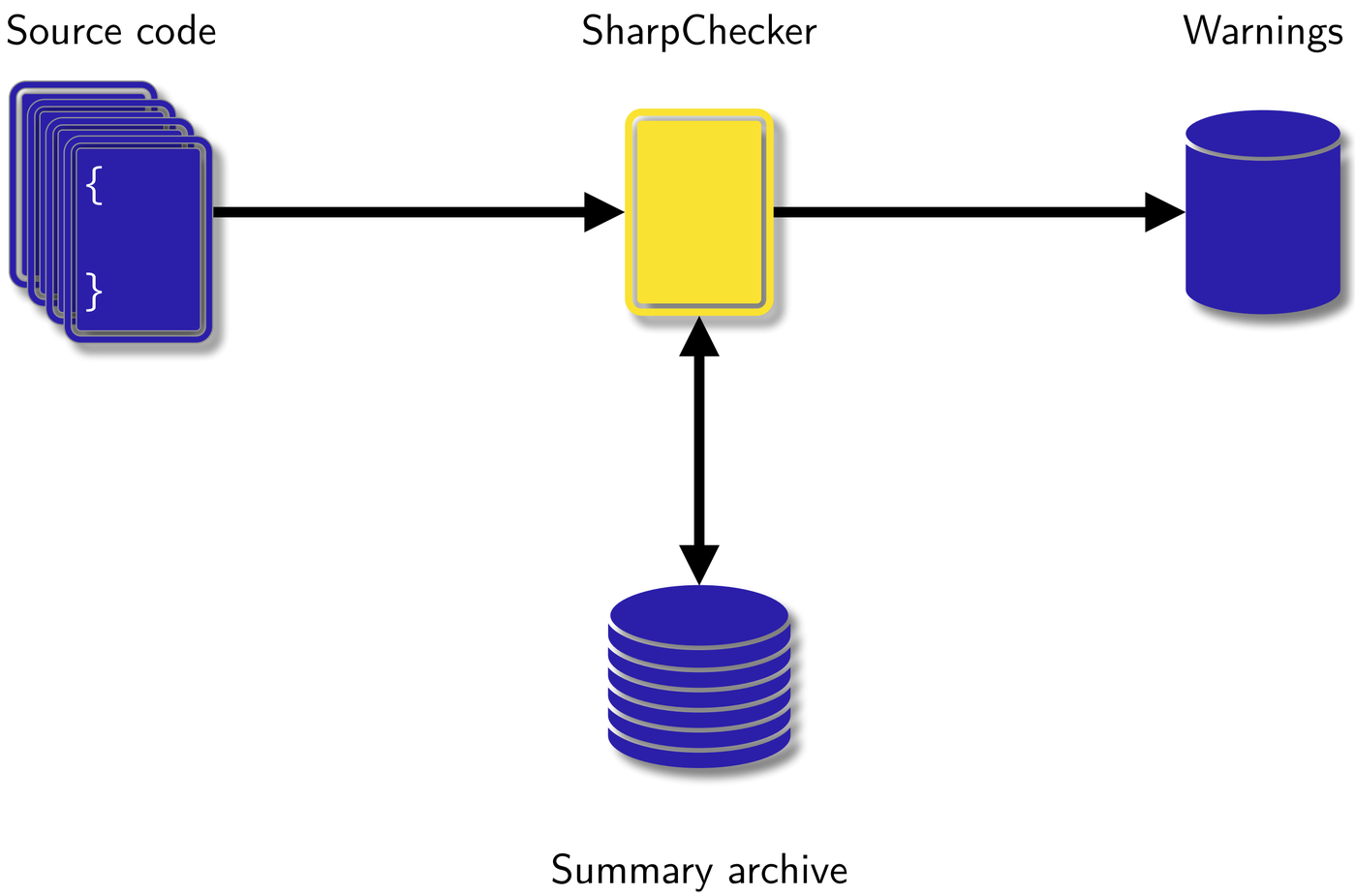
```
public StringBuilder Append(char[] value,
    int startIndex, int charCount)
{
    if (startIndex < 0 || charCount < 0)
        throw new ArgumentOutOfRangeException();

    if (value == null)
    {
        if (startIndex == 0 && charCount == 0)
            return this;
        throw new ArgumentNullException();
    }

    if (charCount > value.Length - startIndex)
        throw new ArgumentOutOfRangeException();

    this._stringValue += value[0];
    return this;
}
```

Code model

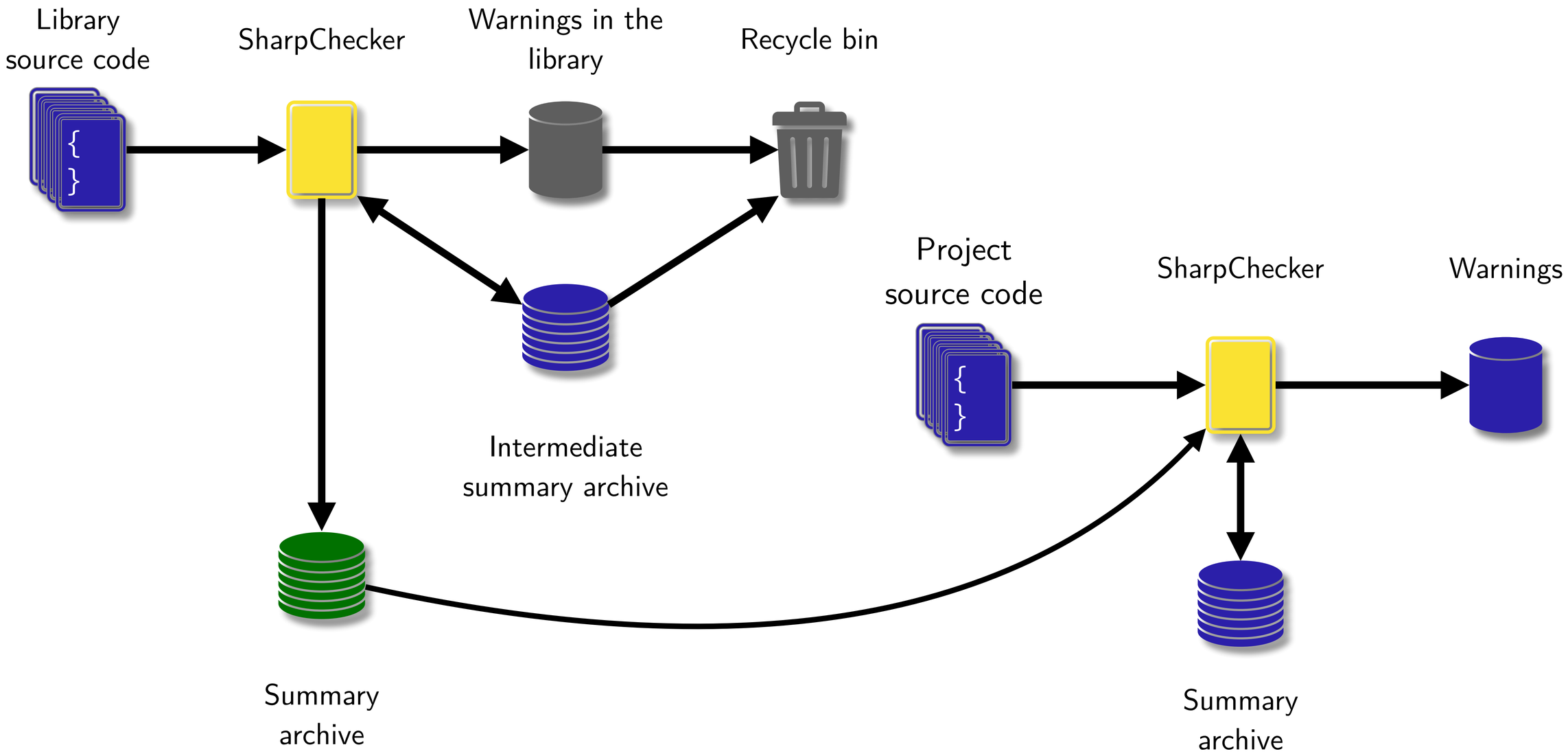
```
public StringBuilder Append(char[] value, int startIndex, int charCount) {
    if (startIndex < 0) {
        throw new ArgumentOutOfRangeException("startIndex", GetRes("GP"));
    }
    if (charCount<0) {
        throw new ArgumentOutOfRangeException("count", GetRes("GP"));
    }

    if (value == null) {
        if (startIndex == 0 && charCount == 0)
            return this;
        throw new ArgumentNullException("value");
    }
    if (charCount > value.Length - startIndex)
        throw new ArgumentOutOfRangeException("count", GetRes("I"));

    if (charCount == 0)
        return this;
    unsafe {
        fixed (char* valueChars = &value[startIndex])
            Append(valueChars, charCount);
    }
    return this;
}
```

Original implementation

Analysis summaries



Library pre-analysis



Comparison of different approaches

	Hardcoded semantics	Property database	Code models	Library pre-analysis
Scalability	10^1	10^4	10^2	10^4
Performance influence	0	1	2	3
Completeness	Yes	No	No	Yes
Maintainability	No	No	No	Yes
Ability to represent unique properties	Yes	No	No	No
Possibility of manual correction	By developers	By users and developers	By users and developers	No
Size	N/A	27 MB	0.8 MB	230 MB
Private code modeling	No	Difficult	Difficult	Yes

Summary size reduction

Initial size: 1.5 GB

Removed:

- Multiple summaries for the same method
- Statistical checkers data
- Private and internal methods
- Unit tests

Result: 230 MB

Analyzer bugs and missing features

Library pre-analysis helps to discover analyzer bugs and imprecisions

- Examples:
  - wrong control flow graph for logical or operator
  - field sensitivity: $a = b \implies a.f = b.f$
- Fixes improve both library and user code analysis

Problem: the analysis with library summaries exceeds limits earlier

Possible solutions:

- condition simplification (both in summaries and in user code)
- different limits for libraries and for user code
- ‘smart’ limits when applying summaries

Evaluation on artificial examples

Without any models – 10 tests don't pass

With analysis summaries – 17 tests don't pass

Reasons:

- summaries don't represent some important unique properties, and currently they override the hardcoded semantics
- the analysis has imprecisions, bugs and performance limits that lead to incorrect or incomplete summaries

Evaluation on open source projects

Quality:

- There are new true positives
- Some true positives disappeared
- Many new false positives appeared

➡ **Further improvements needed**

Performance:

- The analysis became slower by 1 hour (about 60%)

Conclusion

- Library pre-analysis approach has important advantages over other methods
- SharpChecker needs further improvements for practical use of pre-analysis summaries
- The approach can be extended for binary libraries by using a decompiler such as ILSpy