# *Inference of Resource Management Specifications*

Narges Shadab[1], Pritam Gharat[2], Shrey Tiwari[2], Michael D. Ernst[3], Martin Kellogg[4], Shuvendu K. Lahiri[2], Akash Lal[2], Manu Sridharan[1]

[1] University of California, Riverside
[2] Microsoft Research
[3] University of Washington
[4] New Jersey Institute of Technology

# Outline

- Past work overview: *Resource Leak Checker* for Java and C# using resource management specifications
- Inference of Resource Management Specifications
- Evaluation
- Conclusions

# Part I

# Overview: Resource Leak Checker

# What is a Resource Leak?

- In Java and C#, resource management is a shared responsibility of a developer and the runtime environment
- A resource leak occurs when a program fails to free some finite allocated resource after it is no longer needed
- Examples of unmanaged resources: file handles, network sockets, ...
- May lead to
  - Resource starvation
  - System slowdown
  - Whole system crash

# Overview of Past Work: Resource Leak Checker

- We have developed a Resource Leak Checker (RLC) as part of Checker Framework for Java and for C# code using CodeQL
- RLC uses a *light-weight*, *modular*, and sound approach to prevent resource leaks based on checking *resource management specifications*
  - No whole-program alias analysis is required, hence light-weight
- Specifications are written on classes and method boundaries
  - Help RLC to track which objects control a resource and the flow of resources throughout the program
  - Improve the quality of warnings generated by RLC

# Current Work: Inference of Resource Management Specifications

- Inference of resource management specifications
  - Reduce the overhead of developers of manually adding them
  - RLC is now fully automated
- Comparison between hand-written specifications and inferred specifications
- Evaluating RLC with inferred specifications

# Example for RLC

```
1
2  public static SqlConnection getSqlConnection() {
3      var sqlconnection = new SqlConnection(...);
4      ...
5      return sqlconnection;
6  }
7
8  public static void performAction() {
9      SqlConnection connection = getSqlConnection();
10     ...
11     closeConnection(connection);
12 }
13
14 public static void closeConnection(SqlConnection con) {
15     con.Close();
16 }
```

No resource leak in the code snippet

# Example for RLC

```
1  [Owning]
2  public static SqlConnection getSqlConnection() {
3      var sqlconnection = new SqlConnection(...);
4      ...
5      return sqlconnection;
6  }
7
8  public static void performAction() {
9      SqlConnection connection = getSqlConnection();
10     ...
11     closeConnection(connection);
12 }
13
14 public static void closeConnection([Owning] SqlConnection con) {
15     con.Close();
16 }
```

Owning denotes which of the two aliases referring to the same object is responsible for releasing the resource

# Example for RLC

```
1  [Owning]
2  public static SqlConnection getSqlConnection() {
3      var sqlconnection = new SqlConnection(...);
4      ...
5      return sqlconnection; // Obligation transferred
6  }
7
8  public static void performAction() {
9      SqlConnection connection = getSqlConnection();
10     ...
11     closeConnection(connection);
12 }
13
14 public static void closeConnection([Owning] SqlConnection con) {
15     con.Close();
16 }
```

# Example for RLC

```
1  [Owning]
2  public static SqlConnection getSqlConnection() {
3      var sqlconnection = new SqlConnection(...);
4      ...
5      return sqlconnection;
6  }
7
8  public static void performAction() {
9      SqlConnection connection = getSqlConnection();
10     ...
11     closeConnection(connection); // Obligation satisfied
12 }
13
14 public static void closeConnection([Owning] SqlConnection con) {
15     con.Close();
16 }
```

# Example for RLC

```
1  [Owning]
2  public static SqlConnection getSqlConnection() {
3      var sqlconnection = new SqlConnection(...);
4      ...
5      return sqlconnection;
6  }
7
8  public static void performAction() {
9      SqlConnection connection = getSqlConnection();
10     ...
11     closeConnection(connection);
12 }
13
14 public static void closeConnection([Owning] SqlConnection con) {
15     con.Close(); // Obligation satisfied
16 }
```

# Example for RLC

```
1  [Owning]
2  public static SqlConnection getSqlConnection() {
3      var sqlconnection = new SqlConnection(...);
4      ...
5      return sqlconnection;
6  }
7
8  public static void performAction() {
9      SqlConnection connection = getSqlConnection();
10     ...
11     closeConnection(connection);
12 }
13
14 public static void closeConnection([Owning] SqlConnection con) {
15     con.Close();
16 }
```

A warning is reported at the appropriate location (line 9) where the developer should address the issue, rather than at line 3, where the actual resource allocation occurs

# Other Specifications

RLC also uses the following resource management specifications (see paper for more details)

- *MustCall*
- *Calls*
- *MustCallAlias*
- *CreateMustCallFor*

# Specifications Overhead

- RLC expects developers to write specifications
- Annotating legacy code is a huge bottleneck
- Manually incorporating specifications is a multi-step and time-consuming process
  - Took several weeks to annotate a moderately-sized program
  - Unrealistic to expect developers to write these specifications themselves

# Specifications Overhead

- RLC expects developers to write specifications
- Annotating legacy code is a huge bottleneck
- Manually incorporating specifications is a multi-step and time-consuming process
  - Took several weeks to annotate a moderately-sized program
  - Unrealistic to expect developers to write these specifications themselves

Motivated the need for an automatic inference of specifications

# Part II

# Inference of Specifications

# Inference of Resource Management Specifications

- Resource management specifications must capture multiple inter-related properties, including resource ownership, obligations to release, as well as aliasing relationships
- Our inference algorithm employs an *optimistic* approach, identifying resource management specifications that closely align with the developer's likely intentions
- We formalize the inference algorithm as a set of inference rules, such that specifications are inferred by applying the rules to a fixed point
- The inference rules are generic, such that they are implemented in
  - Checker Framework for Java, and
  - CodeQL for C# code
- See paper for more details

# Example: Inference of Specifications I

```
1 void cleanup(Socket socket) throws SocketException {
2   // Disables send and receive on a Socket
3   socket.Shutdown(...); // May throw SocketException
4   socket.Close(); // Closes socket and releases resources
5 }
```

- `cleanup` does not necessarily close `socket`

# Example: Inference of Specifications I

```
1 void cleanup(Socket socket) throws SocketException {
2   // Disables send and receive on a Socket
3   socket.Shutdown(...); // May throw SocketException
4   socket.Close(); // Closes socket and releases resources
5 }
```

- `cleanup` does not necessarily close `socket`
- Reflects only code's behavior
  - No `Owning` specification for `socket`
  - May lead to confusing false positive alarms at call sites of `cleanup`

# Example: Inference of Specifications I

```
1 void cleanup([Owning] Socket socket) throws SocketException {
2   // Disables send and receive on a Socket
3   socket.Shutdown(...); // May throw SocketException
4   socket.Close(); // Closes socket and releases resources
5 }
```

- `cleanup` does not necessarily close `socket`
- Reflects only code's behavior
  - No `Owning` specification for `socket`
  - May lead to confusing false positive alarms at call sites of `cleanup`
- Optimistic approach mirrors the developer's intent
  - Infers `Owning` for `socket`
  - An error is issued by RLC within `cleanup`, exactly where a developer needs to fix the bug

# Example: Inference of Specifications II

```
1  void cleanup([Owning] Socket socket) throws Exception {
2    // Disables send and receive on a Socket
3    socket.Shutdown(...); // May throw SocketException
4    socket.Close(); // Closes socket and releases resources
5  }
```

# Example: Inference of Specifications II

```
1  void cleanup([Owning] Socket socket) throws Exception {
2    // Disables send and receive on a Socket
3    socket.Shutdown(...); // May throw SocketException
4    socket.Close(); // Closes socket and releases resources
5  }
```

$$\textbf{ParamAnnot}(\texttt{Owning}, m, p) \Leftarrow \\ \quad \text{ParamType}(m, T), \\ \quad \textbf{ClassAnnot}(\texttt{MustCall}(m_{pd}), T), \\ \quad \text{Invokes}(s, m, m_{pd}, p, \_)$$

# Example: Inference of Specifications II

```
1  void cleanup([Owning] Socket socket) throws Exception {
2    // Disables send and receive on a Socket
3    socket.Shutdown(...); // May throw SocketException
4    socket.Close(); // Closes socket and releases resources
5  }
```

$$\textbf{ParamAnnot}(\texttt{Owning}, m, p) \Leftarrow \\ \quad \text{ParamType}(m, T), \\ \quad \textbf{ClassAnnot}(\texttt{MustCall}(m_{pd}), T), \\ \quad \text{Invokes}(s, m, m_{pd}, p, \_)$$

Type $T$ of parameter `socket` of $m$ is `Socket` (Resource Type)

# Example: Inference of Specifications II

```
1  void cleanup([Owning] Socket socket) throws Exception {
2    // Disables send and receive on a Socket
3    socket.Shutdown(...); // May throw SocketException
4    socket.Close(); // Closes socket and releases resources
5  }
```

$$\textbf{ParamAnnot}(\texttt{Owning}, m, p) \Leftarrow \\ \quad \text{ParamType}(m, T), \\ \quad \textbf{ClassAnnot}(\texttt{MustCall}(m_{pd}), T), \\ \quad \text{Invokes}(s, m, m_{pd}, p, \_)$$

We model the library type `Socket` with specification `[MustCall("Close")]`

# Example: Inference of Specifications II

```
1  void cleanup([Owning] Socket socket) throws Exception {
2    // Disables send and receive on a Socket
3    socket.Shutdown(...); // May throw SocketException
4    socket.Close(); // Closes socket and releases resources
5  }
```

$$\textbf{ParamAnnot}(\texttt{Owning}, m, p) \Leftarrow \\ \quad \text{ParamType}(m, T), \\ \quad \textbf{ClassAnnot}(\texttt{MustCall}(m_{pd}), T), \\ \quad \text{Invokes}(s, m, m_{pd}, p, \_)$$

There exists an invocation of method `Close` ($m_{pd}$) with receiver object `socket` ($p$) within `cleanup` ($m$)

# Some More Inference Rules*

$\textsc{ClassAnnot}(\texttt{@MustCall}(m_{cd}), C) \leftarrow$ ①
- $\textsc{Method}(m_{cd}, C)$,
- $\neg\textsc{ClassAnnot}(\texttt{@MustCall}(m'_{cd}), C)$,
- $\forall f \in \mathit{OwningFields}(C)$ :
  - $\textsc{FieldDisposal}(f, m_{fd})$,
  - $\textsc{MethodAnnot}(\texttt{@Calls}(f, m_{fd}), m_{cd})$

$\textsc{FieldAnnot}(\texttt{@Owning}, f) \leftarrow$ ②
- $\textsc{Field}(f, C), \textsc{Method}(m, C)$,
- $\textsc{FieldDisposal}(f, m_{fd})$,
- $\textsc{MethodAnnot}(\texttt{@Calls}(f, m_{fd}), m)$

$\textsc{FieldDisposal}(f, m_{fd}) \leftarrow$ ③
- $\textsc{FieldType}(f, T)$,
- $\textsc{ClassAnnot}(\texttt{@MustCall}(m_{fd}), T)$

$\textsc{MethodAnnot}(\texttt{@Calls}(f, m_{fd}), m) \leftarrow$ ④
- $\textsc{FieldDisposal}(f, m_{fd}), \textsc{Invokes}(s, m, m_{fd}, f, \_)$,
- $\textsc{NotWrittenAfter}(f, s, m)$

$\textsc{MethodAnnot}(\texttt{@Calls}(f, m_{fd}), m) \leftarrow$ ⑤
- $\textsc{FieldDisposal}(f, m_{fd})$,
- $\textsc{Invokes}(s, m, m', \texttt{this}, \_)$,
- $\textsc{MethodAnnot}(\texttt{@Calls}(f, m_{fd}), m')$,
- $\textsc{NotWrittenAfter}(f, s, m)$

$\textsc{MethodAnnot}(\texttt{@Calls}(f, m'), m) \leftarrow$ ⑥
- $\textsc{Invokes}(s, m, m', \_, f)$,
- $\textsc{ParamAnnot}(\texttt{@Owning}, m', \_)$,
- $\textsc{NotWrittenAfter}(f, s, m)$

$\textsc{ParamAnnot}(\texttt{@Owning}, m, p) \leftarrow$ ⑦
- $\textsc{ParamType}(m, T)$,
- $\textsc{ClassAnnot}(\texttt{@MustCall}(m_{pd}), T)$,
- $\textsc{Invokes}(s, m, m_{pd}, p, \_)$

$\textsc{ParamAnnot}(\texttt{@Owning}, m, p) \leftarrow$ ⑧
- $\textsc{Invokes}(s, m, m', \_, p)$,
- $\textsc{ParamAnnot}(\texttt{@Owning}, m', \_)$

*See paper for more details

# Part III

# Empirical Evaluation

# Recovering Hand-written Specifications

| | Hand-written Specifications | Inferred Specifications | Percentage (%) |
|---|---|---|---|
| Service 1 | 21 | 21 | 100 |
| Service 2 | 28 | 28 | 100 |
| Service 3 | 24 | 24 | 100 |
| Lucene.Net | 63 | 60 | 95 |
| EF Core | 25 | 17 | 68 |
| Zookeeper | 93 | 66 | 71 |
| Hadoop-hdfs | 91 | 69 | 76 |
| Hbase | 35 | 26 | 74 |

Our algorithm achieved an 82% recovery rate in open-source C# projects, 74% in open-source Java projects, and successfully recovered 100% of the specifications in proprietary C# microservices

# Impact on RLC Warnings I

NS: No Specifications IS: Inferred Specifications

| | #warnings (NS) | #warnings (IS) |
|---|---|---|
| Service 1 | 251 | 240 |
| Service 2 | 45 | 34 |
| Service 3 | 20 | 12 |
| Lucene.Net | 670 | 592 |
| EF Core | 88 | 147 |
| Zookeeper | 138 | 170 |
| Hadoop-hdfs | 26 | 95 |
| Hbase | 828 | 844 |

# Impact on RLC Warnings I

NS: No Specifications IS: Inferred Specifications

| | #warnings (NS) | #warnings (IS) |
|---|---|---|
| Service 1 | 251 | 240 |
| Service 2 | 45 | 34 |
| Service 3 | 20 | 12 |
| Lucene.Net | 670 | 592 |
| EF Core | 88 | 147 |
| Zookeeper | 138 | 170 |
| Hadoop-hdfs | 26 | 95 |
| Hbase | 828 | 844 |

Inference discovers new obligations that RLC does not check in the absence of specifications

# Impact on RLC Warnings I

```
1  [Owning]
2  public static SqlConnection getSqlConnection() {
3      var sqlconnection = new SqlConnection(...);
4      ...
5      return sqlconnection;
6  }
7
8  public static void performAction() {
9      SqlConnection connection = getSqlConnection();
10     ...
11     closeConnection(connection);
12 }
13
14 public static void closeConnection([Owning] SqlConnection con) {
15     con.Close();
16 }
```

Without the `Owning` specification on the return type of `getSqlConnection`, all the calls to this method would likely be overlooked

l Specifications

varnings

| (IS) |
|---|
| 240 |
| 34 |
| 12 |
| 592 |
| 147 |
| 170 |
| 95 |
| 844 |

Inference discovers new obligations that RLC does not check in the absence of specifications

# Impact on RLC Warnings II

| | True Positives | Incorrect Specifications | Missing Specifications |
|---|---|---|---|
| C# | 28% | 0.4% | 25% |
| Java | 19% | 5% | 27% |

The true positive rate using inferred specifications is very close to the rate achieved with hand-written specifications

# Impact on RLC Warnings II

| | True Positives | Incorrect Specifications | Missing Specifications |
|---|---|---|---|
| C# | 28% | 0.4% | 25% |
| Java | 19% | 5% | 27% |

RLC generates no more than 0.4% (C#) and 5% (Java) warnings due to incorrect inferred specifications, indicating that the specifications are generally in line with the developers' intentions

# Impact on RLC Warnings II

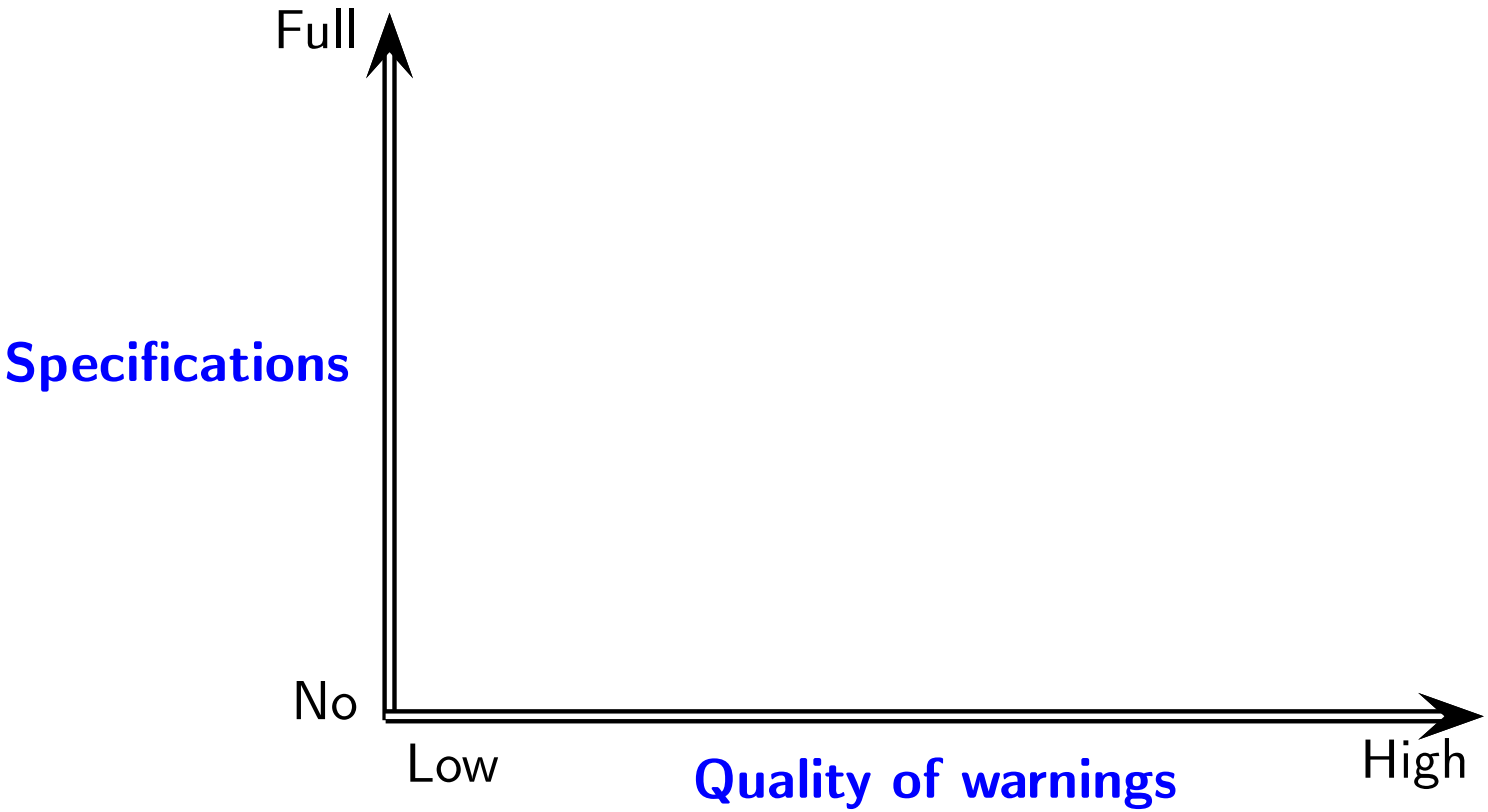
| | True Positives | Incorrect Specifications | Missing Specifications |
|---|---|---|---|
| C# | 28% | 0.4% | 25% |
| Java | 19% | 5% | 27% |

RLC generates no more than 25% (C#) and 27% (Java) warnings due to missing specifications

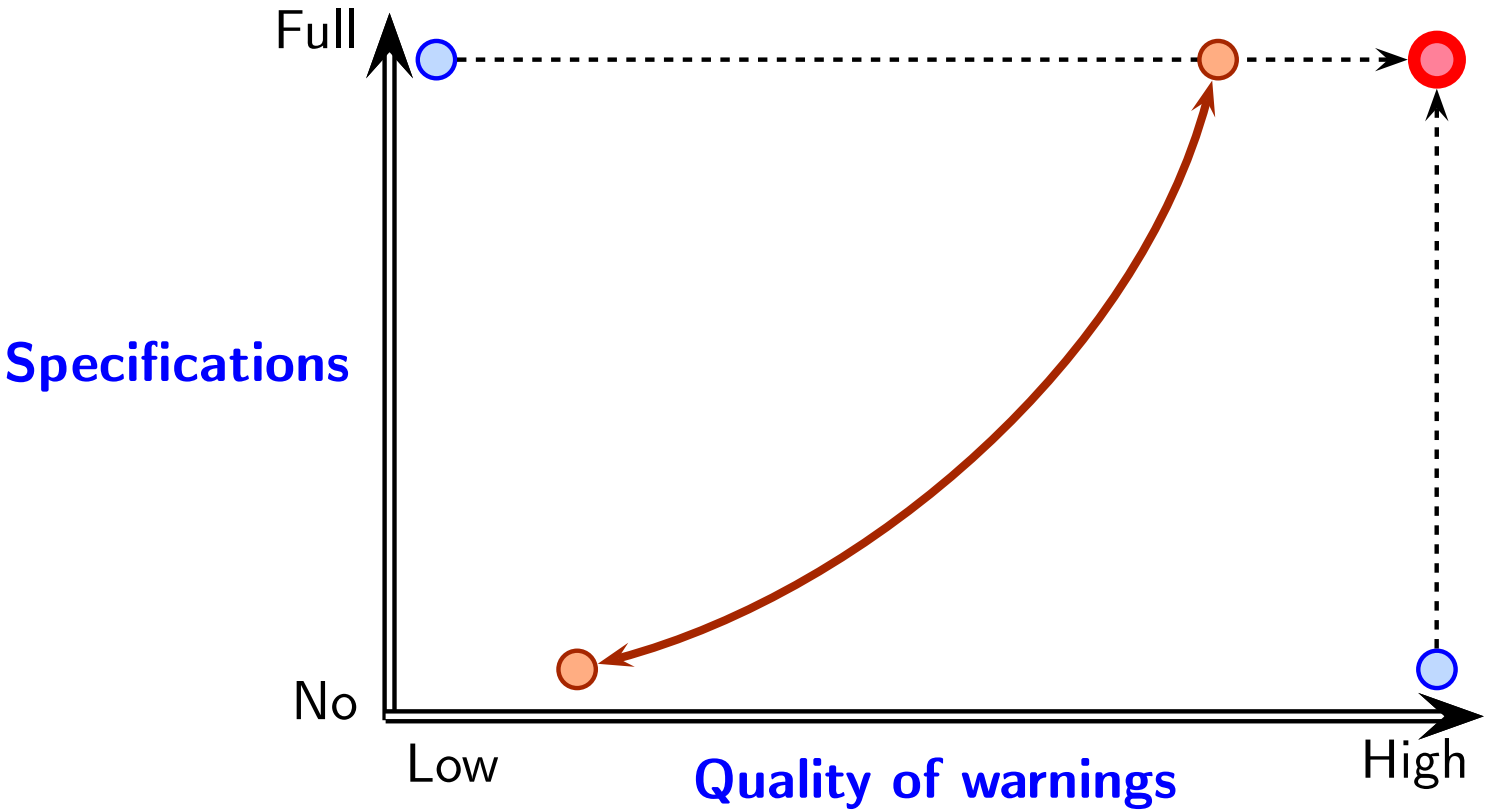
# *Part IV*

# *Conclusions*

# The Role of Specifications for RLC

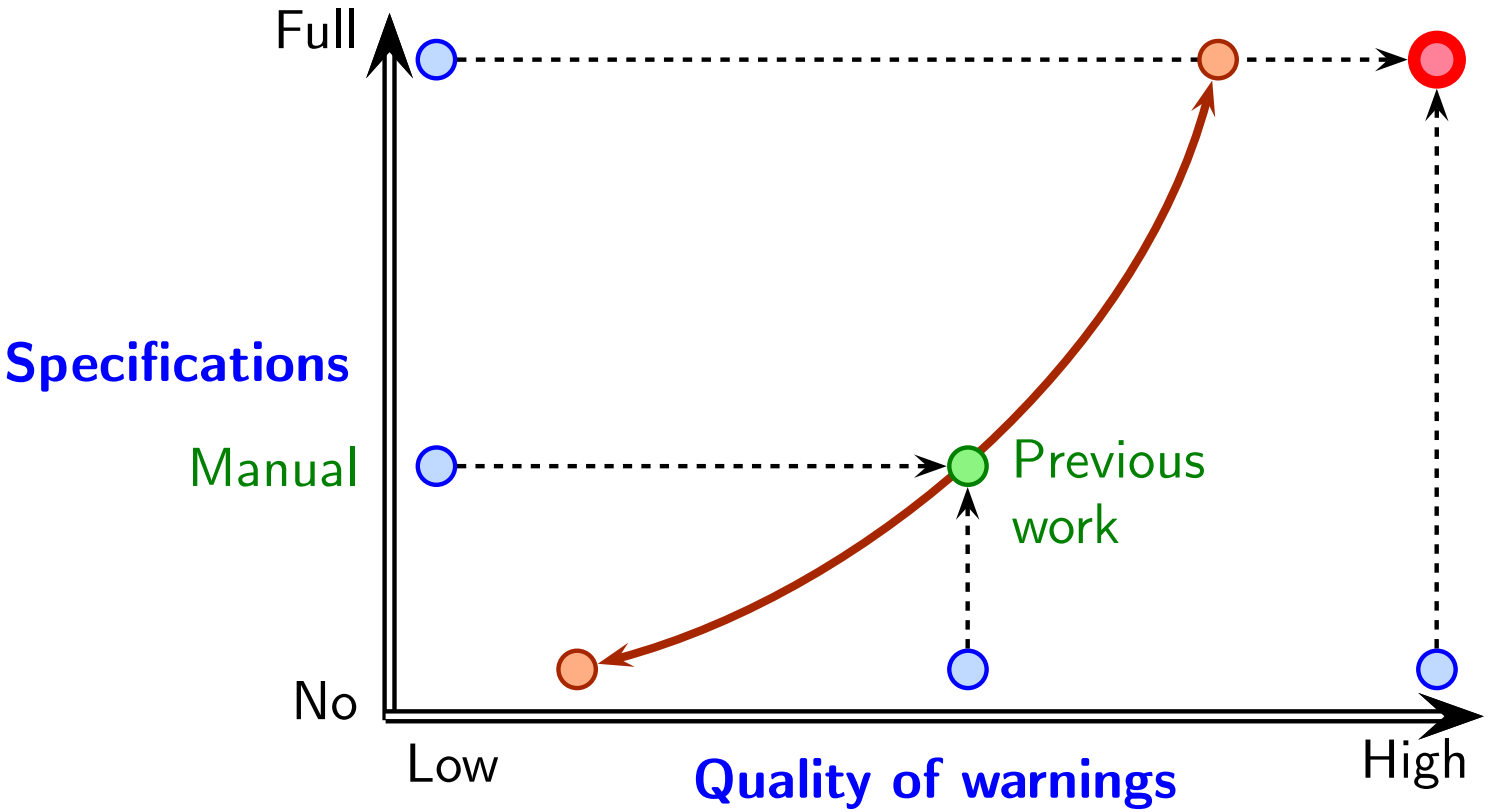


# The Role of Specifications for RLC




RLC with complete specifications might not produce the most accurate results due to inherent imprecision, such as path-insensitivity

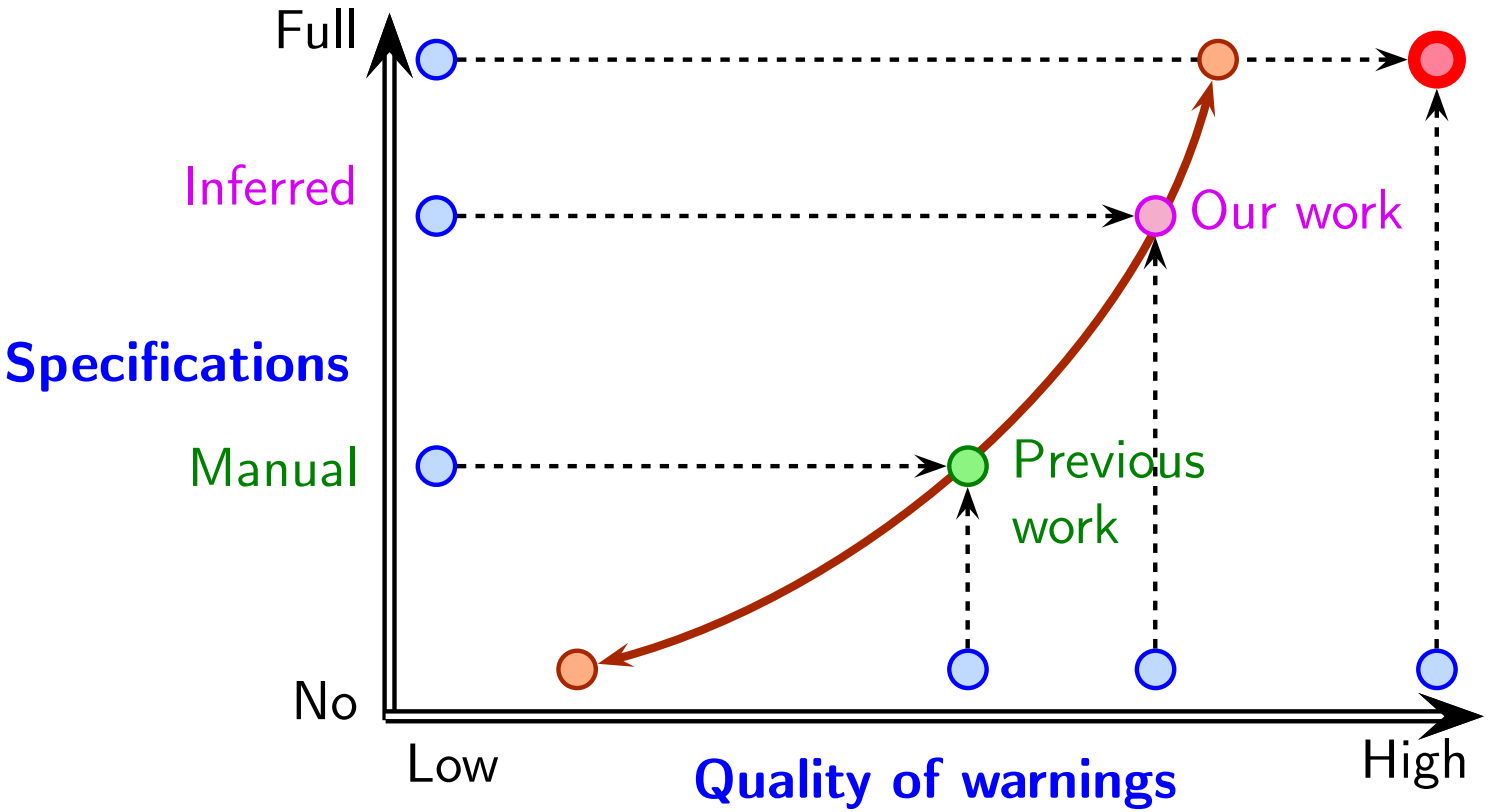
# The Role of Specifications for RLC




Manually written specifications are error prone and effort intensive

We added specifications only for library types

# The Role of Specifications for RLC




Inference of specifications lessens manual work, enabling specifications for all resource types and facilitating the detection of new bugs (6 for Java and 10 for C#)

# Contributions

- Proposed an Inference Algorithm as a set of inference rules for inferring resource management specifications
- Inferred specifications capture developers' intent and generate high quality warnings
- With less manual labor, the average true positive rate for RLC using inferred specifications is almost on par with the rate achieved using manual specifications