Program Analysis

Call Graphs (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2020/2021

Slides adapted from Eric Bodden

Overview

- Introduction
- Single & efficient: CHA, RTA 
- Analyzing assignments: VTA, DTA
- Call graphs and points-to analysis: Spark

Five Algorithms

- **Many algorithms for call graph construction**
  - Class hierarchy analysis (CHA)
  - Rapid type analysis (RTA)
  - Variable type analysis (VTA)
  - Declared type analysis (DTA)
  - General construction framework: Spark

Class Hierarchy Analysis (CHA)

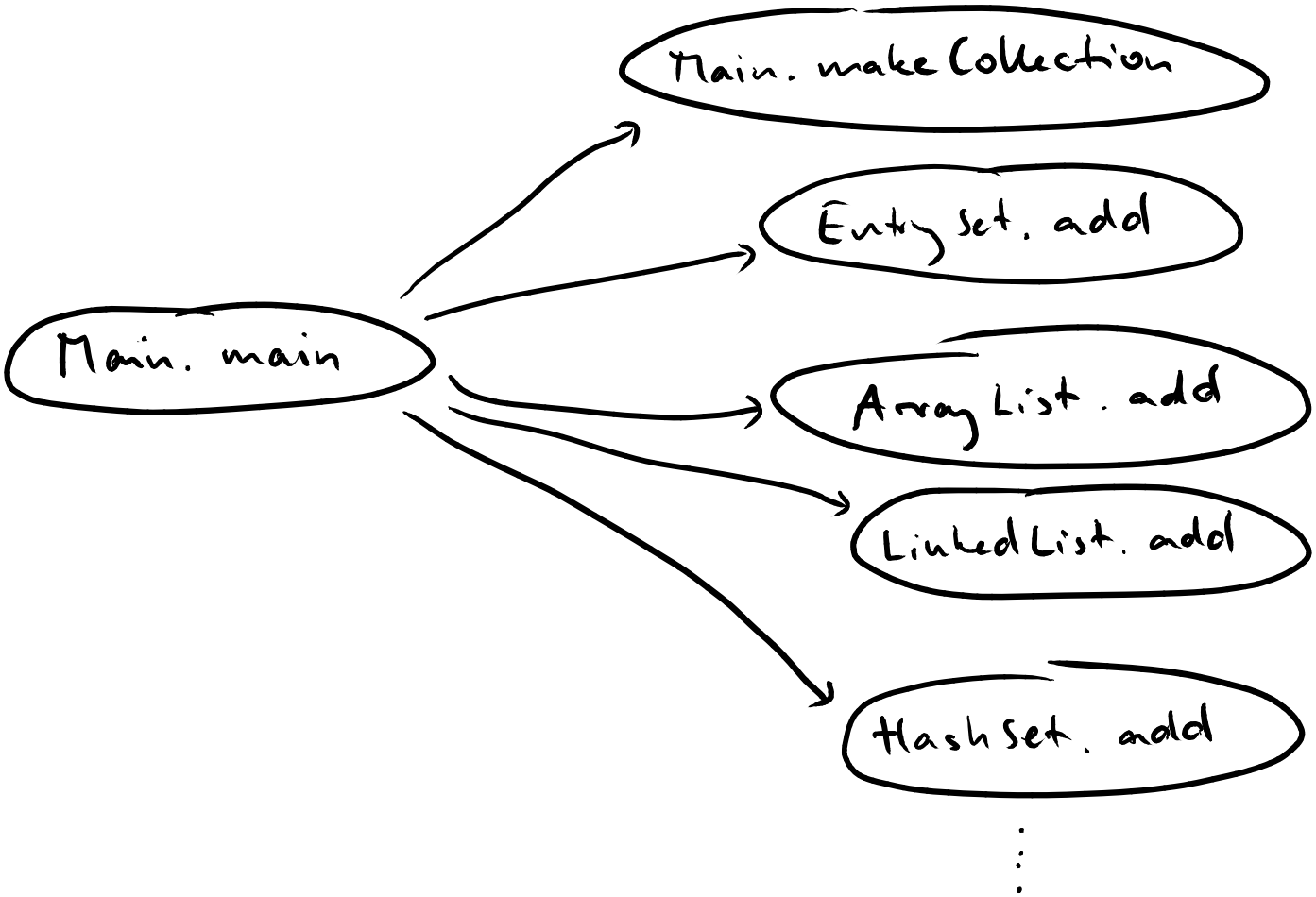
- **Most simple analysis**
- **For a polymorphic call site `m()` on declared type `T`:**
  Call edge to `T.m` and any subclass of `T` that implements `m`

Problem: Polymorphic Calls

```
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Collection c = makeCollection(args[0]);
    c.add("hello");
  }

  static Collection makeCollection(String s) {
    if(s.equals("list")) {
      return new ArrayList();
    } else {
      return new HashSet();
    }
  }
}
```


Main.main
Main.makeCollection
EntrySet.add
ArrayList.add
LinkedList.add
HashSet.add

Class Hierarchy Analysis (CHA)

- **Pros**
  - Very simple
  - Correct: Contains edges for all calls that the program may execute
  - Few requirements: Needs only hierarchy, not other analysis information
- **Cons**
  - Very imprecise: Most edges will never be executed

Rapid Type Analysis (RTA)

- **Like CHA, but:**

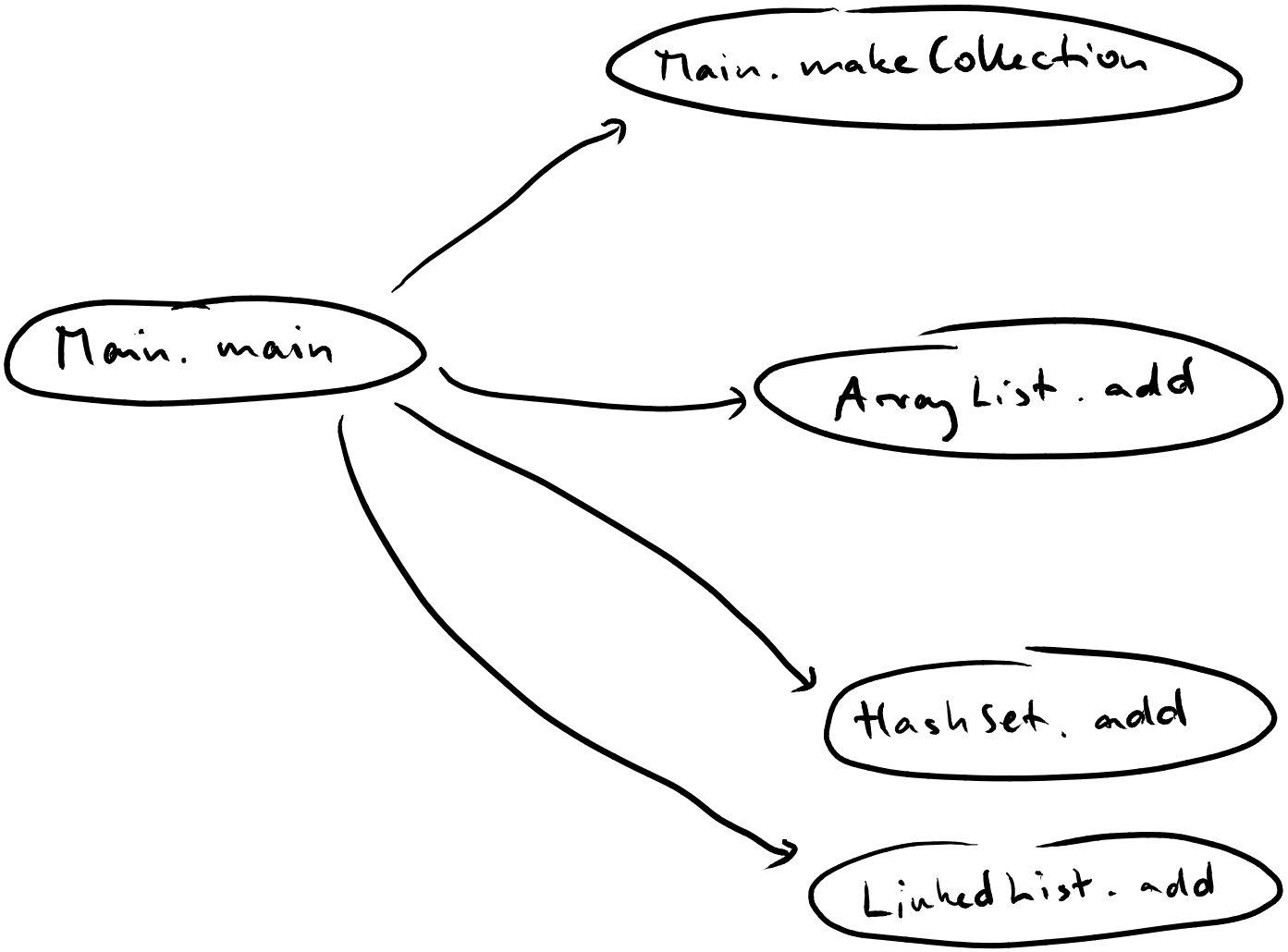
  Take into account only those types that the program actually instantiates

Problem: Polymorphic Calls

```
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Collection c = makeCollection(args[0]);
    c.add("hello");
    new LinkedList();
  }

  static Collection makeCollection(String s) {
    if(s.equals("list")) {
      return new ArrayList();
    } else {
      return new HashSet();
    }
  }
}
```


Main.makeCollection
Main.main
ArrayList.add
HashSet.add
LinkedList.add

Rapid Type Analysis (RTA)

- **Pros**
  - Still pretty fast: Complexity is $\mathcal{O}(|Program|)$
  - Correct
  - Much more precise than CHA:

    Many unnecessary nodes and edges pruned
- **Cons**
  - Doesn't reason about assignments