ChopaChops

Patrick Machado

Contents

Objectives

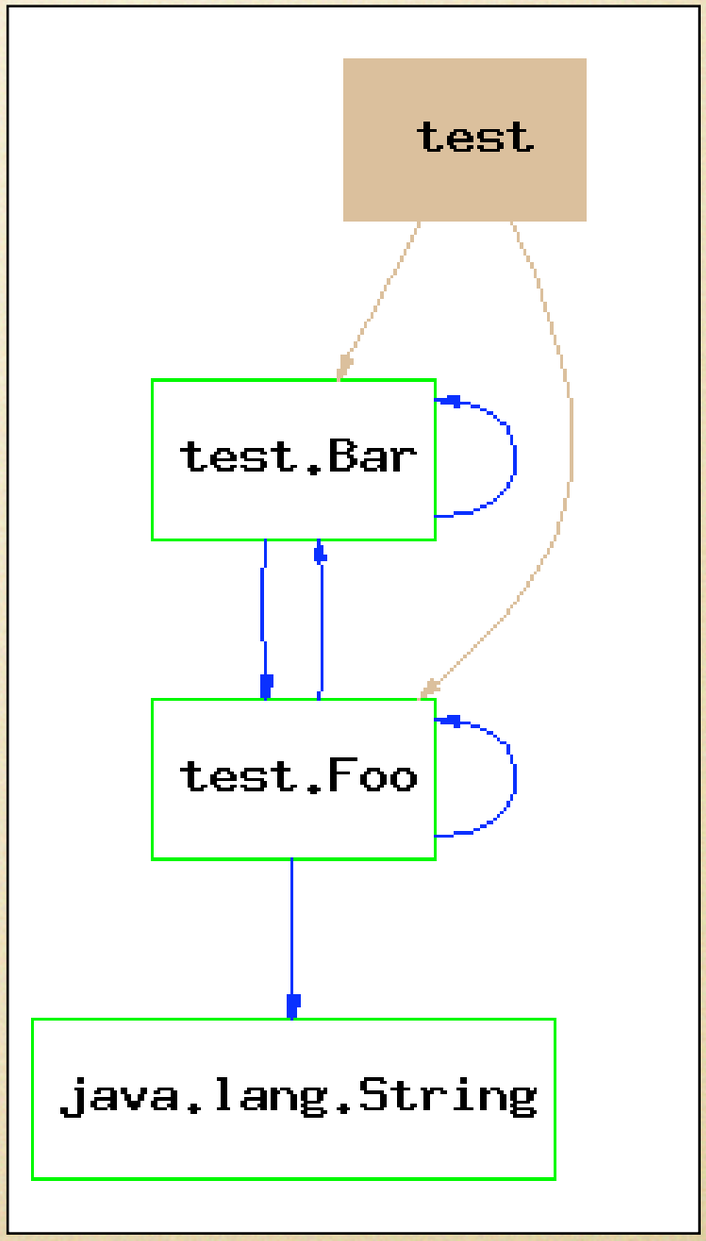
- To derive method invocation relations on Java Programs
- To build graphs
- To compute some graph-based program metrics
- To improve the interactive interface

- Contribute actively to UMinho Haskell Library

Work already done

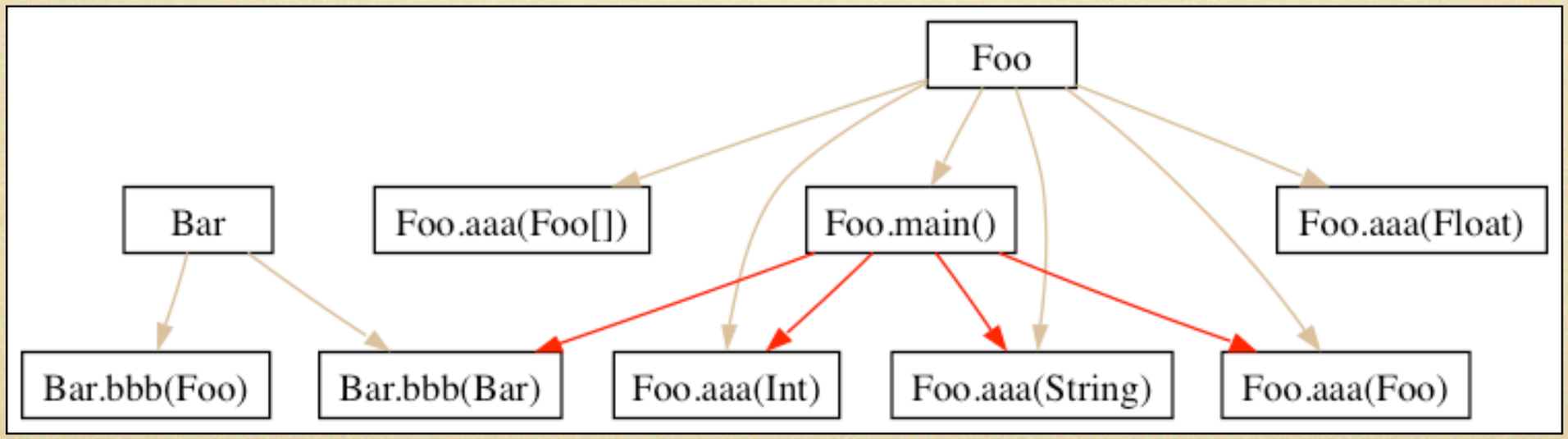
Package Graphs

- Java parsing
- Derivation of package and class-based relations:
  - Imports
  - Inheritance
  - Implementation
  - Nesting
- Visualization of relations in a graph
- Some metrics implementation
- Implementation of slicing and chopping in the graphs



My Contribution
Call Graphs

- Show method invocation relations
- Show method nesting relations



Call Graphs
General

- Without generic programming, it's necessary to handle most of the Java AST
- Used strafunsky to traverse the AST
  - Supports (little) variations on the syntax tree without breaking out
- Built a type inference system to (try to) find out which classes were getting method invocations

Call Graphs

java2ccg

```
type CallGraph
  = LRel CGNode CGNode CGEdgeType

-- | The type of call graph nodes.
data CGNode = CGClass ClassName
            | CGMethod  ClassName MethodName [ParameterType]
...
java2ccg :: (Term a, MonadPlus m) => CGNode -> Declarations -> a -> m CallGraph
java2ccg node@(CGClass cname) decs = ...
java2ccg node@(CGMethod cname mname pars) decs = applyTU $ stop_tdTU worker2
    where
        worker2 = failTU `adhocTU` call2ccg
        call2ccg (NameMethodInvocation ident args) = ...
        call2ccg (SuperMethodInvocation ident args) = ...
        call2ccg (PrimaryMethodInvocation prim ident args)
            = do
                cg <- java2ccg node decs (prim, args)
                let ab = findTypeOf decs cname prim
                let tp = maybe "unknown" (\x -> if (x=="") then "unknown" else x) ab
                let targs = args2str $ findTypeOfArgs decs cname args
                return $ addCallEdge node (CGMethod tp ident targs) cg
```

Call Graphs
Declarations

- Used FiniteMaps to boost lookups

```
type GlobalDeclarations = FiniteMap (ClassName, MemberName) TypeName

type LocalDeclarations = FiniteMap VarName TypeName

type Declarations = (GlobalDeclarations, LocalDeclarations)
```

- API

```
emptyDec :: Declarations

appendDec :: Declarations -> Declarations -> Declarations

addGlobalDec :: Declarations -> (ClassName, MemberName)
    -> TypeName -> Declarations

...

getType :: (MonadPlus m) =>
Declarations -> ClassName -> MemberName -> m TypeName
```

Call Graphs

Type Inference System

```
findTypeOf :: (Term a, MonadPlus m) => Declarations
   -> ClassName -> a -> m TypeName
findTypeOf decs tn = applyTU (once_tdTU worker)
    where
        worker = failTU
            `adhocTU` arrayaccess
            `adhocTU` newcalls
            `adhocTU` methods
...

   arrayaccess (Name_Expression name index) = ...
   arrayaccess _ = mzero

   newcalls (New_comma (ClassOrInterfaceType1 (Name n)) pars)
       = return (name2str n)
   newcalls _ = mzero

   methods (PrimaryMethodInvocation prim ident args) = ...
...
```

Call Graphs

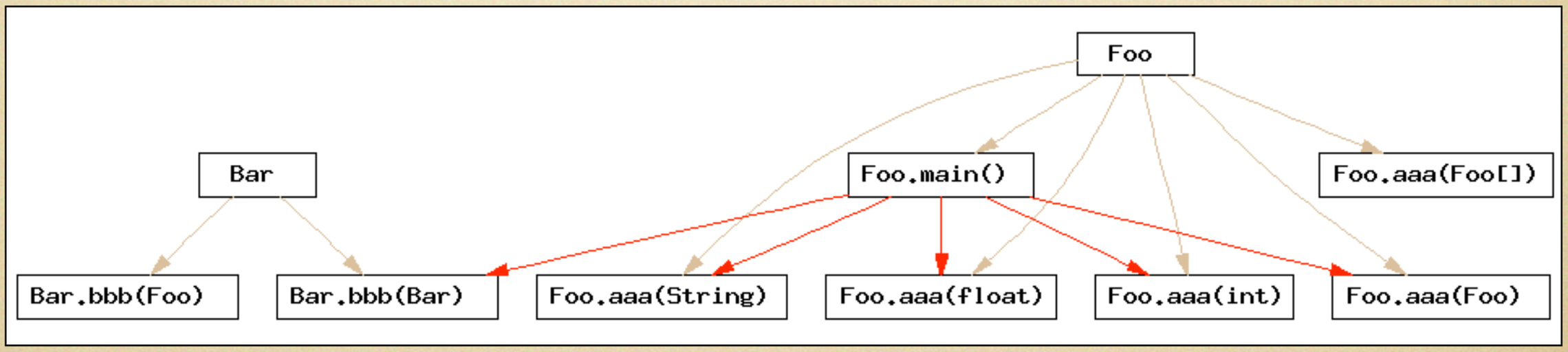
The Algorithm

- Parse the files -> AST
- Traverse AST, collecting type information for methods and fields of the classes
- Traverse AST, building a graph with relations
- Perform slicing and chopping on that graph
- Compute graph metrics
- Print the graph to *dot* notation
- Invoke *GraphViz* to generate an image

Example

```java
01 package test;
02
03 public class Foo
04 {
05     void main()
06     {
07         Foo[] x;
08         Bar d = null;
09         float y;
10         aaa(d.bbb(aaa(aaa(aaa("ola")))
11     }
12     int aaa(String args) { return 1;}
13     Bar aaa(float a) { return null;}
14     float aaa(int x){return null;}
15     Bar aaa(Foo a) {return null; }
16     Bar aaa(Foo[] a) {return null; }
17 }
```

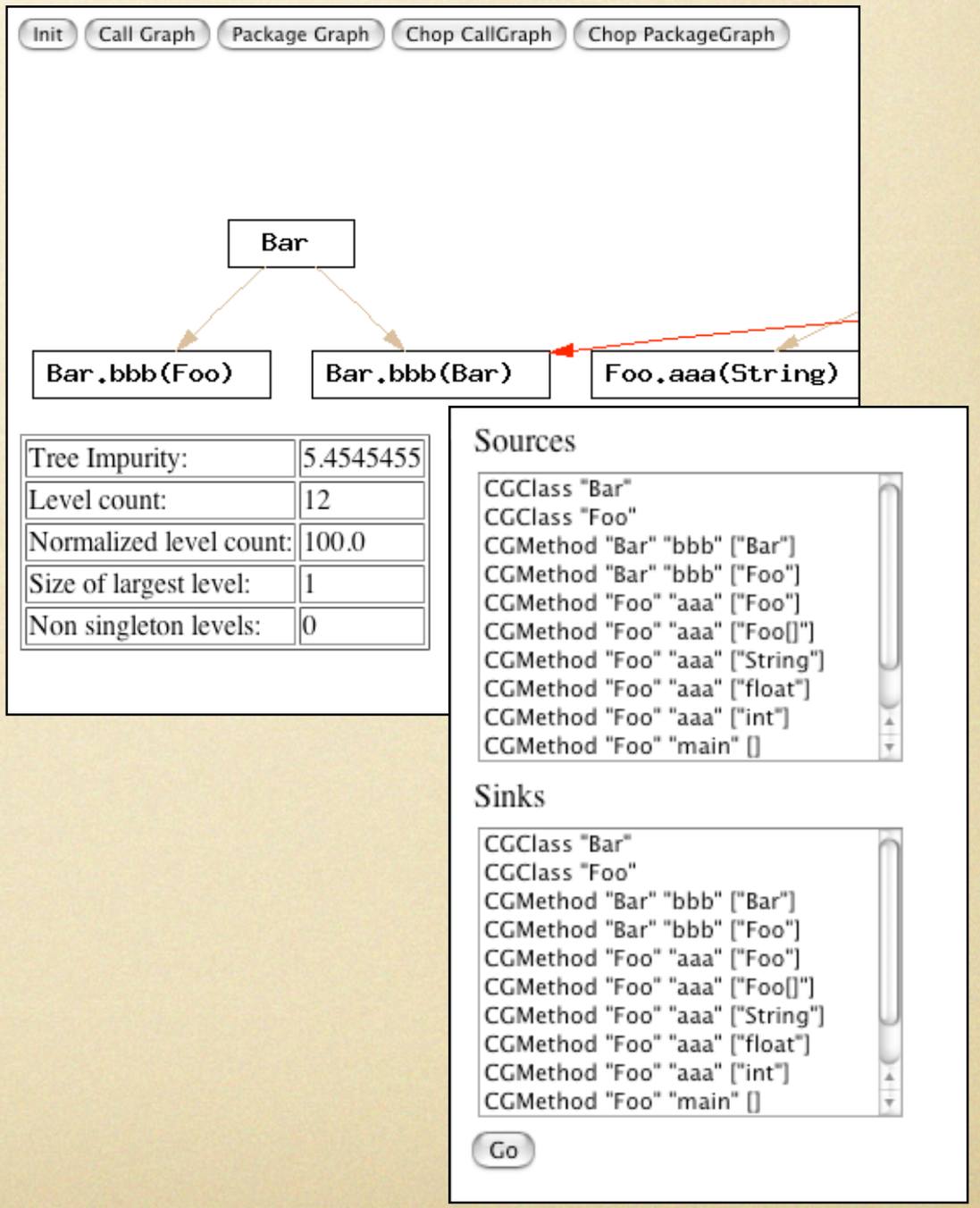
```java
1
2
3 package test;
4
5 class Bar
6 {
7     Foo bbb(Bar x) {return null; }
8     Bar bbb(Foo y) {return null; }
9 }
```



The Tool

ChopaChops Online

- Online Interface using WASH
- Extended *ChopaChopsOnline* to Call Graphs
- Added zip archive functionality
- Added metrics



Problems

- Hard to understand big graphs (real world java programs)
- Type inference system not fully implemented
- Not working with compiled classes
- Does not handle some java features
  - Inheritance / Implementation
  - Static methods
  - Exception handling
  - ...

Real World Application

- Applicable to real-world java applications
- To big programs, it generates graphs not much readable by humans
- Getting more accurate as we support more language features

Future Work
Extension

- Extend the type inference system to support more types
- Compute more graph metrics
- Continue to move to lower level relations:
  - Data flow
  - Control

Conclusion

- Nice progresses towards full program slicing in Java
- At the moment there is no support for many Java features
- **Easy to extend**
- Part of the UHL
  - ChopaChopsOnline (tools)
  - Language.Java.CallGraph
- Much work to do in the future...

Demo