RefaFlex - Safer Refactorings for Reflective Java Programs

Andreas Thies (Fernuni Hagen) and Eric Bodden (TU Darmstadt)

Motivation

- Reflection is a mechanism that allows programs to load and invoke program components through runtime-computed strings
- A refactoring is a program transformation that is meant to preserve the program’s semantics
- Integrated development environments (IDEs) support many refactorings such as renaming or moving classes or members
- Problem: if programs use reflection, all refactorings that current IDEs support are unsound: they are unaware of accesses through reflection

```java
package a;

class Super {
    public int j = 23;
}

public class C extends Super {
    public int i = password();
}

public class Reflection {
    public static void main(String[] args) throws Exception {
        Class<?> c = Class.forName("a.C");
        Field f = c.getField("j");
        System.out.println(f.get(c.newInstance()));
    }
}
```

- The program accesses field C.j through the reflection API
- Initially, this access returns Super.j; renaming C.i to C.j causes C.j (and thus the password) to be returned instead

Constraint-Based Refactoring

- RefaFlex combines two existing tools: TamiFlex [BSS+11] (for monitoring reflective calls) and RefaCola [SKP11], the Refactoring Constraint Language
- RefaCola is a definition language for constraint-based refactorings
- Declarative rules written in RefaCola express the programming language semantics (here Java)
- Concrete refactoring tool: uses these rules as patterns to generate constraints necessary to maintain the program’s semantics
- In RefaFlex such constraints are also generated from runtime data about calls to the reflection API
- Each solution to the constraint system is a valid refactoring

Empirical Evaluation

- applied RefaCola to 21,524 refactorings on 3 open-source projects

Future Work

- Non-functional properties such as security, performance and maintainability are impacted by program changes
- Wish to extend RefaCola with constraints that capture such properties
- Result: users get informed about the impact of program changes with respect to multiple dimensions
- Suggest users alternative but behaviorally equivalent program changes that are optimal with respect to non-functional properties