Type-Safe Requalification

Daniel A. A. Pelsmaeker **Casper Bach**

Automated code refactorings are very useful to change the structure or organization of a code base. Ideally, such refactorings should not change the behavior of the program.

But when refactorings change names or move, introduce, or remove code, it is possible that existing references in the program break or inadvertently point to a different declaration than before the transformation. This would change the behavior of the program.

In this research we show how such references can be fixed by finding *qualifiers* that make the reference resolve to the intended declaration. We call this approach '*requalification*'. and illustrate it here as part of a renaming operation.

```
class A {
  static int x = 1
  class B {
    static int y = 2
    static int z = x + y
    // == 3
  }
}
```

The program on the left has a class `A` with a class `B` lexically nested inside.

Class `A` defines a field `x`, class `B` defines `y` and `z`.

In the expression of field `z` there are two references: one to field `x` in class `A`, and one to field `y` in class `B`.

We want to rename field `y`, to be named `x`.

Rename `y` to `x`

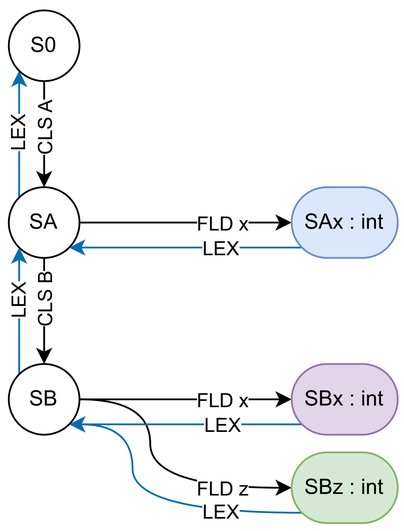
Naively renaming the declaration of `y` in class `B` and all its references to be `x` changes the behavior of the program.

This is because the lexically closer field `x` in class `B` that shadows the field `x` in `A`.

```
class A {
  static int x = 1
  class B {
    static int x = 2
    static int z = x + x
    // == 4
  }
}
```

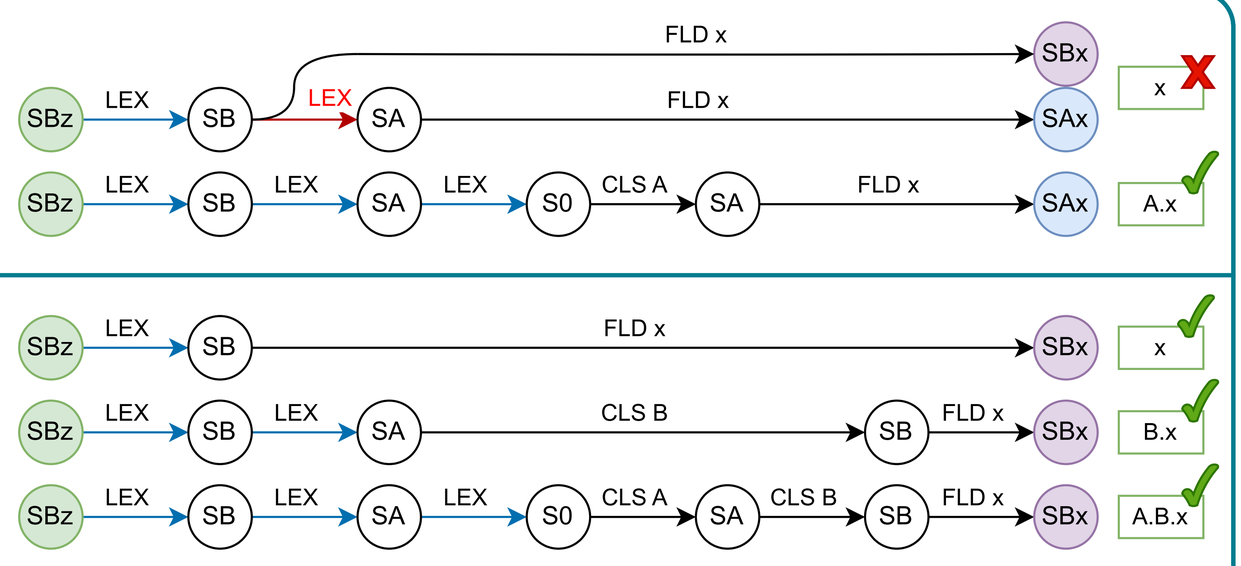
Requalify references

To fix this, we need to 'requalify' the references: finding qualifiers such that the references point to the intended declarations again.



We use the program's scope graph: a model of the program's declarations and scoping structure.

Edges between scopes indicate their relation. Lexically nested scopes are connected by a LEX edge, declarations are connected by a named edge (e.g., CLS A).



We know the scope in which the references occur, SBz, and the scopes to which the references should resolve: SAx and SBx, respectively.

We use the scope graph to find all paths from the source SBz to the target scopes.

```
class A {
  static int x = 1
  class B {
    static int x = 2
    static int z = A.x + x
    // == 3
  }
}
```

For each reference we pick the path that results in the shortest qualified reference.

The result is a program with correct references.

TU Delft