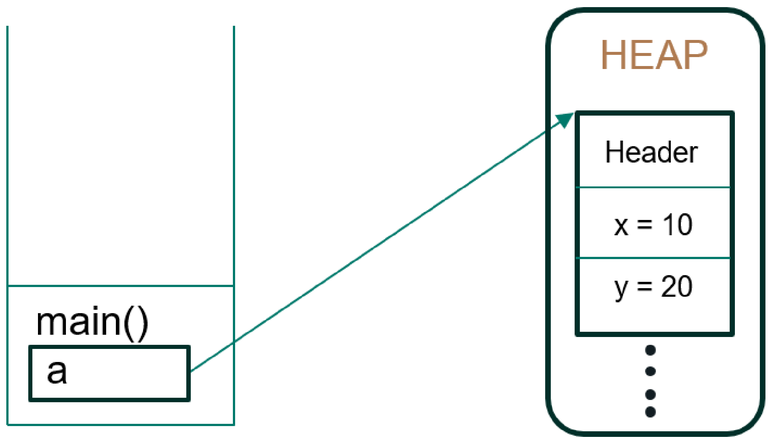
Interprocedural Scalar Replacement

Rohit Singh Yadav, Prof. Manas Thakur
Indian Institute of Technology Bombay

1.Introduction

- Objects in Java are allocated on the heap, and primitive data-types are allocated on stack.
- Heap allocation is considerably slower, is more complex than stack allocation and may also result in allocations scattered all over memory.
- Accessing heap requires dereferencing pointers which is a costly operation.



2. Objective

- Destroy objects and replace them with their members on the stack.
- scalar replacement is an optimization that can decompose an object into its individual components on the stack, most importantly the instance fields of the object.

3. Example

```
main() {
        Obj a;
        a.x = 5;
        a.y = 2;
        bar(a);
        .
        .
    }
bar(Obj a) {
        print(a.x);
    }
```

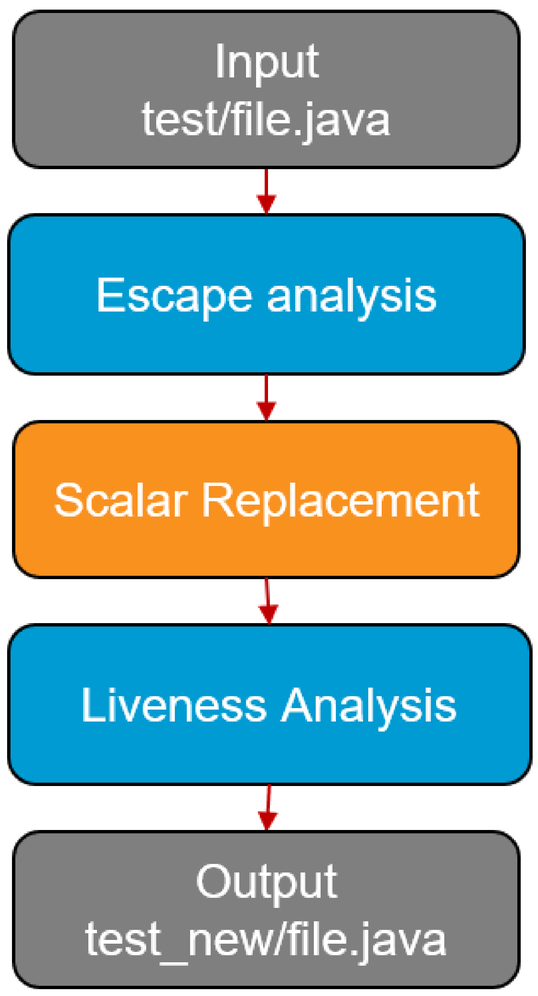
```
main() {
        int a_x = 5;
        int a_y = 2;
        bar(a_x , a_y);
        .
        .
    }
bar(int a_x , int a_y){
        print(a_x);
    }
```

```
main() {
        int a_x = 5;
        int a_y = 2;
        bar(a_x);
        .
        .
    }
bar(int a_x) {
        print(a_x);
    }
```

4. Scalar Replacement

Statement	Condition	Output
`A a = new A();`	***a*** points to a replaceable object	`int a_f;` `int a_g;` `int a_B_f;`
`a.f`	***a*** points to a replaceable object	`a_f`
`a == b`	It can be determined statically that ***a*** and ***b*** can point only to completely different object.	`false`
`((B)a).f`	***a*** points to a replaceable object	`a_B_f`
`a = b`	***a*** points to a replaceable object	`a_f = b_f;` `a_g = b_g;` `a_B_f = b_B_f;`
`a instanceof B`	***a*** points only to objects which are instances of ***B***	`true`

5. Methodology



Escape analysis is performed first, followed by scalar replacement of decomposable objects, then liveness analysis is performed for the fields of object.

source-to source:-

It is easier to implement because we have access to ASTs. Moreover changes are visible. We use JavaParser to access ASTs.