

# Partial Redundancy Elimination

Sebastian Neubauer  
Technische Universität München

March 7, 2019



# Analysis Module

Maybe skip slide

- ▶ Own Java module
- ▶ Implement different analyses
- ▶ Main method takes input file and runs analyses on main function and generates pictures

# How to implement an analysis?

All analyses and transformations are visitors

1. Which type do I want to propagate (domain  $\mathbb{D}$ )?

```
1 public class ReachingDefinitionsAnalysis extends  
    AbstractPropagatingVisitor  
2     <Set<Tupel<Variable, Long>>>
```

# How to implement an analysis?

## 2. Implement methods

*lower upper bound*

*less or equal*

Merge results

Detect if our solution changed

```
1 static Set<Tupel<Variable, Long>> lub(  
2     Set<Tupel<Variable, Long>> s1,  
3     Set<Tupel<Variable, Long>> s2) {  
4     if (s1 == null) return s2;  
5     if (s2 == null) return s1;  
6     HashSet<Tupel<Variable, Long>> res = new HashSet<>();  
7     // Union  
8     res.addAll(s1);  
9     res.addAll(s2);  
10    return res;  
11 }  
12  
13 static boolean lessoreq(Set<Tupel<Variable, Long>> s1,  
14     Set<Tupel<Variable, Long>> s2) {  
15     if (s1 == null) return true;  
16     if (s2 == null) return false;  
17     return s2.containsAll(s1);  
18 }
```

# How to implement an analysis?

## 3. Implement visit for states and interesting transitions

```
1 public Set<Tupel<Variable, Long>> visit(Assignment  
   assignment, Set<Tupel<Variable, Long>>  
   reachingDefinitions) {  
2     Set<Tupel<Variable, Long>> newDefinitions =  
3         new HashSet<>(reachingDefinitions);  
4     if (assignment.getLhs() instanceof Variable) {  
5         Variable variable =  
6             (Variable) assignment.getLhs();  
7         this.processVariableAssignment(newDefinitions,  
            variable, assignment.getDest().getId());  
8     }  
9     return newDefinitions;  
10 }
```

# How to implement an analysis?

4. If the solution converged, return null to end the fix point iteration

```
1 public Set<Tupel<Variable, Long>> visit(State state,  
2     Set<Tupel<Variable, Long>> reachingDefinitions) {  
3     Set<Tupel<Variable, Long>> oldRD = dataflowOf(state);  
4     if (oldRD == null) {  
5         oldRD = new HashSet<>();  
6         dataflowOf(state, reachingDefinitions);  
7         if (reachingDefinitions.equals(oldRD)) {  
8             return reachingDefinitions;  
9         }  
10    }  
11  
12    if (!lessoreq(reachingDefinitions, oldRD)) {  
13        dataflowOf(state,  
14            lub(oldRD, reachingDefinitions));  
15        return lub(oldRD, reachingDefinitions);  
16    }  
17    return null;  
18 }
```

# How to implement an analysis?

All analyses and transformations are visitors

1. Which type do I want to propagate (domain  $\mathbb{D}$ )?
2. Implement methods
  - lower upper bound*      Merge results
  - less or equal*              Detect if our solution changed
3. Implement `visit` for states and interesting transitions
4. If the solution converged, return `null` to end the fix point iteration

# Partial Redundancy Elimination

1. Compute Available Expressions
2. Compute Very Busy Expressions (needed at all paths)
3. Move computations to the earliest possible point



# Partial Redundancy Elimination

1. Compute Available Expressions
2. Compute Very Busy Expressions (needed at all paths)
3. Move computations to the earliest possible point
4. Insert  $T_e = e$  at  $(u, lab, v)$  if  
$$e \in \mathcal{B}[v] \setminus (\llbracket lab \rrbracket_{\mathcal{A}}^{\sharp}(\mathcal{A}[u] \cup \mathcal{B}[u]))$$

Questions?