# Applied Static Analysis

## Monotone Frameworks

Software Technology Group
Department of Computer Science
Technische Universität Darmstadt
Dr. Michael Eichberg

For background information see:

- Principles of Program Analysis; Flemming Nielson, Hanne Riis Nielson, and Chris Hankin; Springer, 2005

Acknowledgements: I would like Dominik Helm for his support in creating the slides!

# Reviewing the Example Analyses

Compare the data flow equations for the four analyses seen so far, e.g.:

Available expressions:

$$AE_{entry}(pc_i) = \begin{cases} \emptyset & \text{if } i = 0 \\ \bigcap \\ AE_{exit}(pc_h) | (pc_h, pc_i) \in flow(S) & otherwise \end{cases}$$

$$AE_{exit}(pc_i) = (AE_{entry}(pc_i) \backslash kill(block(pc_i)) \cup gen(block(pc_i)))$$

Live Variables:

$$LV_{exit}(pc_i) = \begin{cases} \emptyset & \text{if } i \in final(S) \\ \bigcup \\ LV_{entry}(pc_h) | (pc_h, pc_i) \in flow^R(S) & otherwise \end{cases}$$

$$LV_{entry}(pc_i) = (LV_{exit}(pc_i) \backslash kill(block(pc_i)) \cup gen(block(pc_i)))$$

Also if we additionally consider the reaching definitions and very busy expressions analyses we will realize that they are very similar in their general structure. Basically all share the same variation points!

# Generalized Data Flow Equations

These data flow equations can be generalized:

$$Analysis_{\circ}(pc_i) = \begin{cases} \iota & \text{if } i \in E \\ \bigsqcup Analysis_{\bullet}(pc_h) | (pc_h, pc_i) \in F & otherwise \end{cases}$$

$$Analysis*{|bullet}(pc*{i}) = f*{pc_i}(Analysis*{\circ}(pc_{i})) $

with
- $\bigsqcup$ being $\bigcap$ or $\bigcup$

- $F$ either $flow(S)$ or $flow^R(S)$

- $E$ either $\{init(S)\}$ ( `= pc_0` ) or $final(S)$

- $\iota$ being the initial or final analysis information

- $f_{pc_i}$ the transfer function for $pc_i$

---

# Characterization of analyses

- *Forward analyses* use $F = flow(S)$, $\circ = entry$, $\bullet = exit$ and $E = \{init(S)\}$, while
- *Backward analyses* use $F = flow^R(S)$, $\circ = exit$, $\bullet = entry$ and $E = final(S)$

# May and Must analyses

Analyses that require that all paths fulfill a property use $\bigsqcup = \bigcap$ and are called **must analyses**.

Analyses that require at least one path to fulfill a property use $\bigsqcup = \bigcup$ and are called **may analyses**.

# Monotone Framework

- A **monotone framework** consists of
  - a (complete) lattice $L$ that satisfies the *ascending chain condition* where $\bigsqcup$ is the least upper bound and
  - a set $\mathcal{F}$ of **monotone** *transfer functions*
- If the transfer functions are additionally **distributive**, we call it a **distributive framework**

# Instances

- Analyses are **instances** of a monotone framework with
  - the lattice $L$ and transfer functions $\mathcal{F}$ from the framework
  - a flow graph $flow$ that is usually $flow(S)$ or $flow^R(S)$
  - a set of *extremal labels* $E$, typically $\{init(S)\}$ or $final(S)$
  - an *extremal value* $\iota \in L$ for the extremal labels and
  - a mapping $f$ from statements to transfer functions in $\mathcal{F}$

# Transfer Functions from Gen/Kill Functions

The four examples additionally all had their *transfer functions* based on *gen* and *kill* functions:

$$\mathcal{F} = \{f : L \rightarrow L | f(Analysis(pc_i)) = (Analysis(pc_i) \setminus kill(block(pc_i))) \cup gen(block(pc_i))\}$$

# Reviewing the Examples again

- Available expressions
  - $L = \mathcal{P}(ArithExpr)$ with $\bigsqcup = \bigcap$
  - $flow = flow(S)$, $E = \{init(S)\}$ and $\iota = \emptyset$
  - $\bot = ArithExpr$, $\sqsubseteq = \supseteq$

- Reaching definitions
  - $L = \mathcal{P}(Var \times DefSite)$ with $\bigsqcup = \bigcup$
  - $flow = flow(S)$, $E = \{init(S)\}$ and $\iota = \emptyset$
  - $\bot = \emptyset$, $\sqsubseteq = \subseteq$

- Very busy expressions
  - $L = \mathcal{P}(ArithExpr)$ with $\bigsqcup = \bigcap$
  - $flow = flow^R(S)$, $E = final(S)$ and $\iota = \emptyset$
  - $\bot = ArithExpr$, $\sqsubseteq = \supseteq$

- Live variables
  - $L = \mathcal{P}(Var)$ with $\bigsqcup = \bigcup$
  - $flow = flow^R(S)$, $E = final(S)$ and $\iota = \emptyset$
  - $\bot = \emptyset$, $\sqsubseteq = \subseteq$

W.r.t. *reaching definitions* please recall that we assume that we don’t have uninitialized variables.

# Computing a Solution

The so-called *Maximum Fixed Point* solution (MFP) can be computed using the presented worklist algorithm.

# Non-distributive Example: Constant Propagation Analysis

> Determine for each program point, whether or not a variable holds a constant value whenever execution reaches that point.

(*Not every instance of a monotone framework is necessarily distributive.*)

---

## Constant Propagation Analysis - example

```
def m(b : Boolean) : Int = {
  var x =
    if(b)
      -1
    else
      1
  x * x
}
```

Given the constant propagation lattice, it is evident that the application of our transfer function f for `x * x` is not distributive: $f(1 \cup -1) \neq f(-1) \cup f(1)$. The join of -1 and 1 would result in $\top$ and therefore $f(\top) \neq f(-1) \cup f(1)$.

---

# Meet Over All Paths (MOP Solution)

Basic idea:

> Propagate analysis information along paths, then we take the join (or least upper bound) over all paths leading to an elementary block.

Given:

```
if (b) {a = 1; b = 2} else {a = 2, b = 1}
c = a + b
```

The MOP solution would be that c is 3.

---

# MOP Solution for Constant Propagation

**The MOP solution for Constant Propagation is undecidable!**

The proof (out of scope for this lecture) can be done by reduction to the post correspondence problem.

# MFP vs MOP

The MFP solution always safely approximates the MOP solution ($MFP \sqsupseteq MOP$).

However, in case of distributive frameworks the MOP and the MFP solutions coincide.