

## Programmanalyse zum Durchklicken

```
int optimizeMe() {  
    int x = 1;  
    while (random() < 0.5) {  
        x = 2 - x;  
    }  
    return x;  
}  
  
int optimizeMe() {  
    return 1;  
}
```

$\implies$

## Programmanalyse zum Durchklicken

```
int optimizeMe() {                               int optimizeMe() {
  int x = 1;
  while (random() < 0.5) {
  x = 2 - x;                                     ⇒
  }
  return x;                                       return 1;
}                                                 }
```

$$\text{REACH}_{\text{in}}[S] = \bigcup_{p \in \text{preds}[S]} \text{REACH}_{\text{out}}[p]$$

$$\text{REACH}_{\text{out}}[S] = \text{GEN}[S] \cup (\text{REACH}_{\text{in}} - \text{KILL}[S])$$

$$\text{GEN}[d : y \leftarrow f(x_1, \dots, x_n)] = \{d\}$$

$$\text{KILL}[d : y \leftarrow f(x_1, \dots, x_n)] = \text{DEFS}[y] - \{d\}$$

# Programmanalyse zum Durchklicken

Dataflow Analyzer 9000

📁 Datei laden    💾 Datei speichern

```
#define SHIFT 16

int optimizeMe() {
    int x = 1 >> (SHIFT & 0);
    while (random() < 0.5) {
        x = 2 - x;
    }
    return x;
}
```

Verband:  
Zahlen; Flach

Supremum:  
Vereinigung  $\cup$

Vorwärtsanalyse  
 Rückwärtsanalyse

Analysereihenfolge:  
Worklist

Control flow graph illustrating dataflow analysis for the provided code:

- [Start]** (d1: x = 1): Gen: {d1}, Kill: {d2}. Edge to while: x=⊥, Edge to return: x=1.
- while (random() < 0.5)**: Gen: ∅, Kill: ∅. Edge to loop body: x=⊥, Edge to return: x=⊥.
- d2: x = 2 - x**: Gen: {d2}, Kill: {d1}. Edge to while: x=⊥, Edge to return: x=⊥.
- return x [End]**: Gen: ∅, Kill: ∅. Edge to return: x=⊥.

Führe Schritt aus    Führe bis Fixpunkt aus

⏪ ————— ⏩

# Programmanalyse zum Durchklicken

Dataflow Analyzer 9000

📁 Datei laden    💾 Datei speichern

```
#define SHIFT 16

int optimizeMe() {
    int x = 1 >> (SHIFT & 0);
    while (random() < 0.5) {
        x = 2 - x;
    }
    return x;
}
```

Verband: Zahlen; Flach

Supremum: Vereinigung  $\cup$

Vorwärtsanalyse  
 Rückwärtsanalyse

Analysereihenfolge: Worklist

Gen: {d1}    Kill: {d2}

[Start]  
d1: x = 1

Gen:  $\emptyset$     Kill:  $\emptyset$

while (random() < 0.5)

Gen: {d2}    Kill: {d1}

d2: x = 2 - x

Gen:  $\emptyset$     Kill:  $\emptyset$

return x  
[End]

Führe Schritt aus    Führe bis Fixpunkt aus

# Programmanalyse zum Durchklicken

Dataflow Analyzer 9000

📁 Datei laden    💾 Datei speichern

```
#define SHIFT 16

int optimizeMe() {
    int x = 1 >> (SHIFT & 0);
    while (random() < 0.5) {
        x = 2 - x;
    }
    return x;
}
```

Verband: Zahlen; Flach

Supremum: Vereinigung  $\cup$

Vorwärtsanalyse  
 Rückwärtsanalyse

Analysereihenfolge: Worklist

Gen: {d1}    Kill: {d2}

[Start] d1: x = 1

Gen: ∅    Kill: ∅

while (random() < 0.5)

Gen: {d2}    Kill: {d1}

d2: x = 2 - x

Gen: ∅    Kill: ∅

return x [End]

Führe Schritt aus    Führe bis Fixpunkt aus

# Programmanalyse zum Durchklicken

Dataflow Analyzer 9000

📁 Datei laden    💾 Datei speichern

```
#define SHIFT 16

int optimizeMe() {
    int x = 1 >> (SHIFT & 0);
    while (random() < 0.5) {
        x = 2 - x;
    }
    return x;
}
```

Verband:  
Zahlen; Flach

Supremum:  
Vereinigung  $\cup$

Vorwärtsanalyse  
 Rückwärtsanalyse

Analysereihenfolge:  
Worklist

Gen: {d1}    Kill: {d2}

[Start]  
d1: x = 1

Gen: ∅    Kill: ∅

while (random() < 0.5)

Gen: {d2}    Kill: {d1}

x=1    d2: x = 2 - x    x=⊥

Gen: ∅    Kill: ∅

return x  
[End]

Führe Schritt aus    Führe bis Fixpunkt aus

# Programmanalyse zum Durchklicken

Grober Umriss:

- ▶ Interesse an Compilern, Optimierungen
- ▶ **Java** empfohlen, aber kein Zwang
- ▶ Empfehlungen für Bibliotheken, aber kein Zwang

# Programmanalyse zum Durchklicken

Grober Umriss:

- ▶ Interesse an Compilern, Optimierungen

(Keine Angst, wir erklären's euch)

- ▶ **Java** empfohlen, aber kein Zwang
- ▶ Empfehlungen für Bibliotheken, aber kein Zwang