



# Kombinierte Datenfluss-Analysen

---

Hauptseminar Programmanalysen

12.Juni.2008

Fridolin Häuser



# Überblick

---

- Allgemeines
- Sequentielle Kombination
- Kombination in einem Algorithmus
- Kombination über ein modulares System
- Interprozedural



# Analysetechniken

---

- Konstanten Propagierung
- Elimination von totem Code
- Inlining
- Klassenanalyse



# Notation

---

- Unterschiede in der Notation
- Der Meet-Operator:

$$\text{any} + \perp = \text{any}$$

$$\text{any} + \top = \top$$

$$C + C = C$$

$$C_1 + C_2 = \top \text{ if } C_1 \neq C_2$$



# Überblick

---

- Allgemeines
- Sequentielle Kombination
- Kombination in einem Algorithmus
- Kombination über ein modulares System
- Interprozedural



# Sequentielle Kombination

---

- Analysen werden nacheinander ausgeführt
- Einmalig oder Iterativ
- Monotonität und Terminierung





# Sequentielle Analyse - Beispiel

---

```
X := 10;           ([X = 10])
Y := 10;           ([X = 10], [Y = 10])
if (Y == 10) {
    DoSomething();
} else {
    DoSomethingElse();
}
```



# Sequentielle Analyse - Beispiel

---

```
X := 10;           ([X = 10])
Y := 10;           ([X = 10], [Y = 10])
if (true) {       ([X = 10], [Y = 10])
    DoSomething();
} else {
    DoSomethingElse();
}
```



# Sequentielle Analyse - Beispiel

---

X := 10;	R
Y := 10;	R
if (true) {	R
DoSomething();	R
} else {	
DoSomethingElse();	U
}	



# Sequentielle Analyse - Beispiel

---

X := 10; R

Y := 10; R

DoSomething(); R



# Phase-Ordering Problem

---

- Optimierungsproblem
- Bewirkt unterschiedliche Ergebnisse
- Enorme Anzahl an Permutationen
- Heuristische Problemlösung



# Beispiel

---

```
X := 10;                ([X=10])
while (...) {          ([X=10])
  if (X == 10) {
    ...
  } else {
    ...
    X := X + 1;
  }
}
Y := X;
```



# Beispiel

---

<code>X := 10;</code>	<code>([X=10])</code>
<code>while (...) {</code>	<code>([X=10])</code>
<code>if (X == 10) {</code>	<code>([X=⊥])</code>
<code>...</code>	
<code>} else {</code>	
<code>...</code>	
<code>X := X + 1;</code>	
<code>}</code>	
<code>}</code>	<code>([X=⊥])</code>
<code>Y := X;</code>	<code>([X=⊥], [Y=⊥])</code>



# Beispiel

---

- Konstanten Propagierung und Elimination von totem Code verändern den Programmcode nicht
- Fehlendes Zusammenspiel der Analysen



# Fazit zur sequentiellen Kombination

---

- Einfache Umsetzung
- Phase-Ordering Problem
- Kein direktes Zusammenspiel zwischen den Analysen
- Iterativ: lange Compilezeit



# Überblick

---

- Allgemeines
- Sequentielle Kombination
- Kombination in einem Algorithmus
- Kombination über ein modulares System
- Interprozedural



# Algorithmische Kombination

---

- Kombination verschiedener Analysen zu einem Algorithmus
- Individuelle Erstellung
- Sparse Conditional Constant von Wegman und Zadeck



# Sparse Conditional Constant

---

Elemente:

- SSA-Graph
- Zwei Arbeitslisten  
(FlowWorkList/SSAWorkList)
- Wahrheitswert zu jeder Programmfluss-Kante
- Verbund



# Beispiel

---

```
X := 10;  
while (...) {  
    if (X == 10) {  
        ...  
    } else {  
        ...  
        X := X + 1;  
    }  
}  
Y := X;
```



# Laufzeit

---

- SSA-Kante maximal zwei Besuche
- Programmflussknoten maximal einen Besuch pro Eingangskante im Graphen
- $O(\text{SSA-Kanten} * \text{CFG-Kanten})$



# Fazit zum kombinierten Algorithmus

---

- Gute Laufzeiten
- Ermöglicht direktes Zusammenspiel der Analysen
- Schlechte Modularität



# Überblick

---

- Allgemeines
- Sequentielle Kombination
- Kombination in einem Algorithmus
- Kombination über ein modulares System
- Interprozedural



# Kombination über ein System

---

Am Beispiel von Lerner, Grove und Chambers

Ziele:

- Modularität
- Zusammenspiel der Analysen
- Gute Compilezeit



# Lerner, Grove und Chambers

---

- SSA-Form
- Replacement Graph
- Graphtransformationen
- Monotonität und Terminierung der Gesamtfunktion



# Beispiel

---

- Elimination toten Codes
- Konstanten Propagierung



# Beispiel

---

```
X := 10;  
while (...) {  
    if (X == 10) {  
        ...  
    } else {  
        ...  
        X := X + 1;  
    }  
}  
Y := X;
```



# Beispiel zur Modularität

---

- Elimination toten Codes
- Konstanten Propagierung
- Inlining
- Klassenanalyse



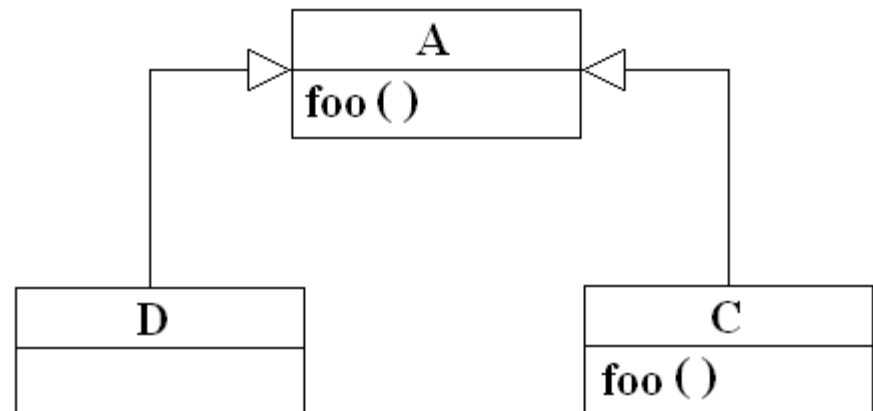
# Beispiel - Klassendefinition

---

```
class A {  
    function foo(): A {return new A;}  
}  
class C extends A {  
    function foo(): A {return self;}  
}  
class D extends A {}
```

# Beispiel

```
Decl X: A;  
X := new C;  
While (...) {  
  decl B: Bool;  
  B := X instanceof C;  
  if (B) {  
    X := X.foo();  
  } else {  
    X := new D();  
  }  
}
```





# PICK - Funktion

---

Welche Transformation wird genommen, wenn mehrere Analysen den Graphen verändern wollen?

-> PICK - Funktion



# Experimentelle Laufzeit

---

Zeilenzahl	kombiniert	L/G/C	seq. Iterativ	seq.
msort(110)	1.00	0.99	1.01	6.28
	1.00	1.11	6.04	0.20
richard(400)	1.00	1.00	1.07	13.66
	1.00	1.18	6.54	1.03
typechecker (20000)	1.00	1.01	1.01	5.30
	1.00	1.18	6.55	0.94
compiler (50000)	1.00	1.02	1.00	4.05
	1.00	1.15	7.46	1.22



# Fazit zu den Frameworks

---

- Gute Modularität
- Terminierung und Monotonität
- Ordentliches Laufzeitverhalten
- Ermöglicht das Zusammenspiel der Analysen
- PICK – Funktion



# Überblick

---

- Allgemeines
- Sequentielle Kombination
- Kombination in einem Algorithmus
- Kombination über ein modulares System
- Interprozedural



# Interprozedural

---

- Optimierung mit interprozeduralen Informationen
- Cloning
- Inlining



# Zusammenfassung

---

- Sequentielle Kombination
- Algorithmische Kombination
- Kombination über ein Framework

Eigenschaften:

- Modularität
- Laufzeit
- Zusammenspiel der Analysen