

# Vorlesung Software-Reengineering

Dipl. Inf. Gunther Vogel<sup>1</sup>

<sup>1</sup>ISTE

Fakultät für Informatik, Elektrotechnik und Informationstechnik  
Universität Stuttgart  
(Prof. Dr. Rainer Koschke, Universität Bremen)

Wintersemester 2005/06

# Überblick I

# Mustersuche

## Lernziele

- Varianten der Mustersuche abhängig von der Programmdarstellung
- Formalismen zur Mustersuche

## Kontext

- Mustersuche ist Teil von Code-Transformationen

# Anwendungen der Mustersuche

- Lokalisierung bei Code-Transformationen
- Programmiermuster (Planerkennung):
  - Jahr-2000-Problematik
    - Suche nach Datumsmanipulationen
    - Suche nach Schaltjahrberechnung
  - Erkennung typischer Datenstrukturen bzw. Algorithmen, wie z.B. Suche nach Listeniteration
  - Code-Anomalien oder Verletzung von Coding-Style-Guides

- Text
- Lexeme
- Syntax
- (statisch-)semantische Information
- globale Daten- und Kontrollfluss-Information

# Problem

Die Suche nach Klone hat ergeben, dass es unzählige Tausch-Operationen der Form

```
t = a; a = b; b = t;
```

gibt. Ersetzen Sie sie gegen `SWAP(&a,&b)` mit:

```
void SWAP (int *x, int *y) {  
    int t = *x;  
    *x = *y;  
    *y = t;  
}
```

- Suche nach Worten wie "‘tausch’", "‘swap’" etc.

# Textuelle Ebene

- Suche nach Worten wie " 'tausch' ", " 'swap' " etc.
- Suche nach regulärem Ausdruck:

```
grep
```

```
'[a-zA-Z][a-zA-Z0-9]* *=[a-zA-Z][a-zA-Z0-9]* *; *  
[a-zA-Z][a-zA-Z0-9]* *=[a-zA-Z][a-zA-Z0-9]* *; *  
[a-zA-Z][a-zA-Z0-9]* *=[a-zA-Z][a-zA-Z0-9]* *; *' file.c
```

# Textuelle Ebene

- Suche nach Worten wie " 'tausch' ", " 'swap' " etc.
- Suche nach regulärem Ausdruck:

```
grep  
'[a-zA-Z][a-zA-Z0-9]* *=[a-zA-Z][a-zA-Z0-9]* *; *  
[a-zA-Z][a-zA-Z0-9]* *=[a-zA-Z][a-zA-Z0-9]* *; *  
[a-zA-Z][a-zA-Z0-9]* *=[a-zA-Z][a-zA-Z0-9]* *; *' file.c
```

- Suche nach " 'kontextsensitivem regulären' " Ausdruck:

```
grep  
'\([a-zA-Z][a-zA-Z0-9]*\) *=[a-zA-Z][a-zA-Z0-9]*\) *; *  
\2 *=[a-zA-Z][a-zA-Z0-9]*\) *; *  
\3 *=[a-zA-Z][a-zA-Z0-9]*\) *; *' file.c
```

Gegenbeispiele:

```
t = a; a = b;  
b = t;
```

```
t=a; a=b; /* ein kommentar */ b /* noch einer */ =t;
```

# Lexem-basierte Suche I

Tokenstrom:

swap-match () =  $\langle id = id; id = id; id = id; \rangle$

Mit zusätzlichen Token-Attributen:

$\langle id_1 = id_2; id_3 = id_4; id_5 = id_6; \rangle$

**where**  $id_1.text = id_6.text$

**and**  $id_2.text = id_3.text$

**and**  $id_4.text = id_5.text$

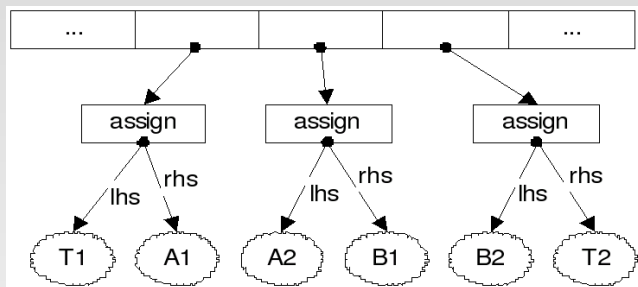
# Lexem-basierte Suche II

Gegenbeispiele:

```
t = a; a = b; ; b = t;
```

```
t=a[i]; a[i]=b[j]; b[j]=t;
```

# Syntax-basierte Suche I



mit den folgenden Gleichheiten (nicht Identitäten):

$$T1 = T2, A1 = A2 \text{ und } B1 = B2$$

seq (\_, assign (\$T, \$A), assign (\$A, \$B), assign (\$B, \$T), \_)

# Syntax-basierte Suche II

Gegenbeispiel:

```
float t, a, b;
```

...

```
t = a; a = b; b = t;
```

wegen Signatur von SWAP:

```
SWAP (int *x, int *y)
```

# Statisch-semantische Information I

```
swap-match ($T, $A, $B) =  
seq (-,  
  assign ($T, $A),  
  assign ($A, $B),  
  assign ($B, $T),  
-)
```

**where** type (\$T)=type(\$A)=type(\$B)=int

# Statisch-semantische Information II

Gegenbeispiel:

```
t = a; a = b; x = y; b = t;
```

Erster Lösungsansatz:

```
seq (_, assign ($T, $A), _ ,  
      assign ($A, $B), _ ,  
      assign ($B, $T), _)
```

Aber:

```
t = a; a = b; goto l; b = t;
```

# Statisch-semantische Information III

Zweiter Lösungsansatz:

```
swap-match ($T, $A, $B) =  
  seq (_, assign ($T, $A), $X ,  
        assign ($A, $B), $Y ,  
        assign ($B, $T), _)
```

```
where not contains ($X, goto)  
and not contains ($X, label)  
and not contains ($Y, goto)  
and not contains ($Y, label)
```

Gegenbeispiel:

```
t=a; goto l; x=y; l: a=b; b=t;
```

(... wobei l von keinem anderen goto angesprungen wird.)

# Kontrollflussinformation I

swap-match (\$T, \$A, \$B) =

S1: assign (\$T, \$A),

S2: assign (\$A, \$B),

S3: assign (\$B, \$T),

**where** pdom\* (S1) = S2

**and** pdom\* (S2) = S3

**and** dom\* (S2) = S1

**and** dom\* (S3) = S2

Gegenbeispiele:

```
t = a; a = b; t = x; b = t;
```

```
a = 1; b = 2;
```

```
t = a; a = b; b = t;
```

- Ergebnis: `a==t==b==2`, wenn `t` und `a` Aliase sind.
- Ergebnis von `SWAP (&a, &b)`:  
`a == 2` und `b == 1`

# Datenflussinformation I

swap-match ( $T, A_0, B_0, A_1, B_1$ ) =

S1: assign ( $T, A_0$ ),

S2: assign ( $A_1, B_0$ ),

S3: assign ( $B_1, T$ )

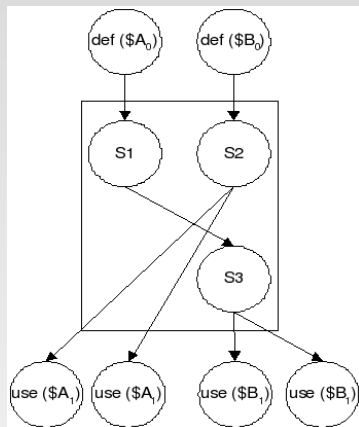
**where**

S1 verwendet nur  $def(A_0)$

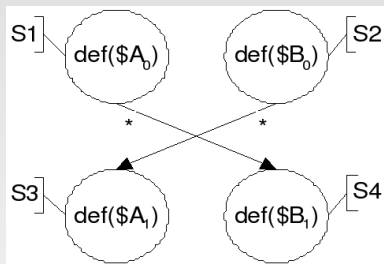
**and** S2 verwendet nur  $def(B_0)$

**and** S3 verwendet nur Definition in S1

**and** Def. in S1 wird nur in S3 verwendet



# Datenflussinformation II



uses (S3) = {def (\$B0)}  
**and** uses (S4) = {def (\$A0)}

- Text: reguläre Ausdrücke über Zeichenketten

# Zusammenfassung der Formalismen

- Text: reguläre Ausdrücke über Zeichenketten
- Lexeme: reguläre Ausdrücke über Tokens mit Bedingungen über Token-Attribute

# Zusammenfassung der Formalismen

- Text: reguläre Ausdrücke über Zeichenketten
- Lexeme: reguläre Ausdrücke über Tokens mit Bedingungen über Token-Attribute
- Syntax: Matching über abstrakten Syntaxbaum

# Zusammenfassung der Formalismen

- Text: reguläre Ausdrücke über Zeichenketten
- Lexeme: reguläre Ausdrücke über Tokens mit Bedingungen über Token-Attribute
- Syntax: Matching über abstrakten Syntaxbaum
- (statisch-)semantische Information: Matching über abstrakten Syntaxbaum mit statisch-semantischen Bedingungen

# Zusammenfassung der Formalismen

- Text: reguläre Ausdrücke über Zeichenketten
- Lexeme: reguläre Ausdrücke über Tokens mit Bedingungen über Token-Attribute
- Syntax: Matching über abstrakten Syntaxbaum
- (statisch-)semantische Information: Matching über abstrakten Syntaxbaum mit statisch-semantischen Bedingungen
- globale Daten- und Kontrollfluss-Information:

# Zusammenfassung der Formalismen

- Text: reguläre Ausdrücke über Zeichenketten
- Lexeme: reguläre Ausdrücke über Tokens mit Bedingungen über Token-Attribute
- Syntax: Matching über abstrakten Syntaxbaum
- (statisch-)semantische Information: Matching über abstrakten Syntaxbaum mit statisch-semantischen Bedingungen
- globale Daten- und Kontrollfluss-Information:
  - Matching über abstrakten Syntaxbaum mit statisch-semantischen Bedingungen und Bedingungen über Kontroll- und Datenfluss

# Zusammenfassung der Formalismen

- Text: reguläre Ausdrücke über Zeichenketten
- Lexeme: reguläre Ausdrücke über Tokens mit Bedingungen über Token-Attribute
- Syntax: Matching über abstrakten Syntaxbaum
- (statisch-)semantische Information: Matching über abstrakten Syntaxbaum mit statisch-semantischen Bedingungen
- globale Daten- und Kontrollfluss-Information:
  - Matching über abstrakten Syntaxbaum mit statisch-semantischen Bedingungen und Bedingungen über Kontroll- und Datenfluss
  - Matching über Kontroll- bzw. Datenflussgraph

# Wiederholungs- und Vertiefungsfragen I

- Worauf kann sich die Mustersuche stützen?
- Gegeben ist das folgende (unvollständige) Programmfragment zur Berechnung der Tage eines Jahres: ...
- Gib ein Muster zur Erkennung eines Programmiermusters zur Bestimmung der Wochentage im Februar (Schaltjahr muss beachtet werden), das sich auf informelle Information stützt.
- Welche Vor- und Nachteile hat dieses Muster?
- Gib ein Muster an, das sich auf lexikalische Information stützt.
- Vorteile/Nachteile?
- Gib ein Syntax-Muster an:
- Welche Vor- und Nachteile hat dieses Muster?
- Gib ein Muster an, das Kontrollfluss berücksichtigt.
- Was ist Kontrollabhängigkeit?

# Wiederholungs- und Vertiefungsfragen II

- Welche Vor- und Nachteile hat dieses Muster?
- Wie könnte man das obige Muster erweitern, so dass man garantieren kann, dass die Variable "year" sich nicht zwischen den Tests verändert?
- Wie kann man ein solches Muster implementieren, wenn als Zwischendarstellung ein abstrakter Syntaxbaum mit SSA-Form verwendet wird?

