

Abstrakte Interpretation



Hauptseminar

„Testen der Sicherheitseigenschaften der JavaCard“

Vortrag am 24. Januar 2002

Bernhard Liebl

Um was es geht...



- Betrachtung der abstrakten Bedeutung von Programmen
- Entkopplung von konkreten Eingabewerten oder Flußverhalten
- Treffen allgemein gültiger Aussagen mittels mathematischer Beweise

Motivation I



- Abstraktes Debugging
- Beweis von Programmverhalten
- Zusicherung von Sicherheitseigenschaften

Motivation II



- Erzeugung optimierten Codes in Compilern
- Konstantenpropagierung, Lebende Variablen
- Toter Code, Unerreichbarer Code
- Compile-Zeit Array-Bound-Checking, Überlaufprüfung für Typkonvertierungen

Beispiel I



```
void debuggingExample()
{
    int T[ 100 ];
    int n;

    n = readInteger();

    for( int i = 0; i < n; i++ )
        T[ i - 1 ] = readInteger();

    ...
}
```

Beispiel I

```
void debuggingExample()
```

```
{
```

```
    int T[ 100 ];
```

```
    int n;
```

```
    n = readInteger();
```

☹ **n > 101**

```
    for( int i = 0; i < n; i++ )
```

```
        T[ i - 1 ] = readInteger();
```

☹ **i = 0**

```
        ...
```

```
}
```

Beispiel II



```
void anotherDebuggingExample()
{
    int i;

    i = readInteger();
    j = 2 * i + 1;

    assert( ( j % 2 ) != 0 );
}
```

Beispiel II

```
void anotherDebuggingExample()
{
    int i;

    i = readInteger();
    j = 2 * i + 1;

    assert( ( j % 2 ) != 0 );
}
```

😊 immer wahr

Beispiel III



```
bool checkNumber( int k )
{
    assert( k >= 0 and k < 10 );
    int i = 3, j = 0;

    while( j < i )
    {
        k = k + j;
        j = j + 1;
    }

    return k < 14;
}
```

Beispiel III

```
bool checkNumber( int k )
{
    [k → ? ]
    assert( k ≥ 0 and k < 10 );
    [ k → [0;9] ]
    int i = 3, j = 0;
    [ i → { 3 }, j → { 0 }, k → [0;9] ]

    while( j < i )
    ...
}
```

Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
    k = k + j;
```

```
    [ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
}
```

...

Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
    k = k + j;
```

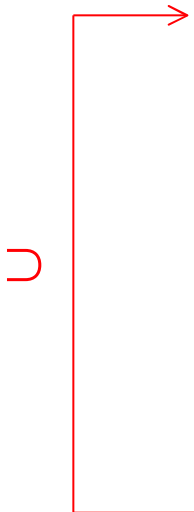
```
    [ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
}
```

...



Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1 }, k → [0;9] ] ∩ { j | j < i }
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
    k = k + j;
```

```
    [ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
}
```

...

Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
    k = k + j;
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;10] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
```

```
}
```

...

Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1, 2 }, k → [0;9] ]
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
    k = k + j;
```

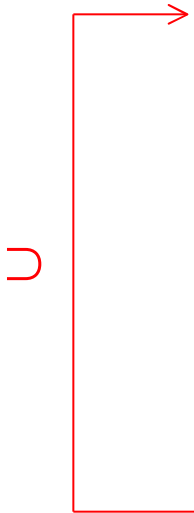
```
    [ i → { 3 }, j → { 0, 1 }, k → [0;10] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
```

```
}
```

...



Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1, 2 }, k → [0;9] ] ∩ { j | j < i }
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;9] ]
```

```
    k = k + j;
```

```
    [ i → { 3 }, j → { 0, 1 }, k → [0;10] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
```

```
}
```

...

Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1, 2 }, k → [0;9] ]
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
```

```
    k = k + j;
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;12] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
```

```
}
```

...

Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;9] ]
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
```

```
    k = k + j;
```

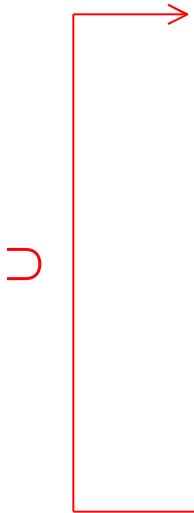
```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;12] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
```

```
}
```

...



Beispiel III

...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;9] ] ∩ { j | j < i }
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
```

```
    k = k + j;
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;12] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
```

```
}
```

...

Beispiel III



...

```
int i = 3, j = 0;
```

```
[ i → { 3 }, j → { 0 }, k → [0;9] ]
```

```
[ i → { 3 }, j → { 0, 1, 2 }, k → [0;9] ]
```

```
while( j < i )
```

```
{
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
```

```
    k = k + j;
```

```
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;12] ]
```

```
    j = j + 1;
```

```
    [ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
```

```
}
```

...

Beispiel III

...

int i = 3, j = 0;

$[i \rightarrow \{3\}, j \rightarrow \{0\}, k \rightarrow [0;9]]$

$[i \rightarrow \{3\}, j \rightarrow \{0, 1, 2\}, k \rightarrow [0;9]]$

while(j < i)

{

$[i \rightarrow \{3\}, j \rightarrow \{0, 1, 2\}, k \rightarrow [0;10]]$

k = k + j;

$[i \rightarrow \{3\}, j \rightarrow \{0, 1, 2\}, k \rightarrow [0;12]]$

j = j + 1;

$[i \rightarrow \{3\}, j \rightarrow \{0, 1, 2, 3\}, k \rightarrow [0;12]]$

}

...

Fixpunkt

Beispiel III



```
...
while( j < i )
{
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
    k = k + j;
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;12] ]
    j = j + 1;
    [ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
}

[ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
return k < 14;
}
```

Beispiel III

```
...
while( j < i )
{
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;10] ]
    k = k + j;
    [ i → { 3 }, j → { 0, 1, 2 }, k → [0;12] ]
    j = j + 1;
    [ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
}
```

```
[ i → { 3 }, j → { 0, 1, 2, 3 }, k → [0;12] ]
```

```
return k < 14; 😊 immer wahr
```

```
}
```

Formalisierung

Konkrete
Maschine
KM

z \longrightarrow z'

Abstrakte
Maschine
AM

\underline{z} \longrightarrow $\underline{z'}$

Zustandsübergang

Kontrollpunkte



$[c_0]$

$a = 5; [c_1]$

$b = 2; [c_2]$

$a = a + b; [c_3]$

Umgebungen



: IDENT

: $\{ a_1 \mapsto v_1, \dots, a_n \mapsto v_n \}$

' = $[\cdot]$

Umgebungen

: IDENT

: $\{ a_1 \mapsto v_1, \dots, a_n \mapsto v_n \}$

' = [\cdot]

$0 =$

$1 = [a=5] \quad 0$

$2 = [b=2] \quad 1$

$3 = [a=a+b] \quad 2$

$[c_0]$

$a = 5; [c_1]$

$b = 2; [c_2]$

$a = a + b; [c_3]$

Umgebungen

: IDENT

: $\{ a_1 \mapsto v_1, \dots, a_n \mapsto v_n \}$

' = $[\cdot]$

$0 = \{ a, b \}$

$1 = \{ a = 5, b \}$

$2 = \{ a = 5, b = 2 \}$

$3 = \{ a = 7, b = 2 \}$

$[c_0]$

$a = 5; [c_1]$

$b = 2; [c_2]$

$a = a + b; [c_3]$

Datenflußgleichung I



$$x_1 = f_1(x_1, \dots, x_n)$$

$$x_2 = f_2(x_1, \dots, x_n)$$

⋮

⋮

⋮

$$x_n = f_n(x_1, \dots, x_n)$$

Datenflußgleichung II

$$P = \{1, \dots, n\}$$

$$F_P: P \rightarrow P$$
$$F_P = f_1, \dots, f_n$$

Programmbeschreibung

$$P = F_P(P)$$

Datenflußgleichung

Programmbedeutung

Finde (kleinstes) P , so daß

$$P = F_P(P)$$

$\langle 1, \dots, n \rangle$ ist Fixpunkt von $F_P = \langle f_1, \dots, f_n \rangle$

$$f_i(\langle 1, \dots, n \rangle) = \langle 1, \dots, i, \dots, n \rangle$$

„Naive“ Iteration



Berechne in jedem Iterationsschritt F^j

$$x_i^{j+1} = f_i(x_1^j, \dots, x_n^j) \text{ f\u00fcr jedes } 1 \leq i \leq n$$

Chaotische Iteration

Berechne in jedem Iterationsschritt F^j

$$F^j(\begin{matrix} x_1, \dots, x_n \end{matrix}) = \begin{matrix} x_1, \dots, x_{i-1}, \\ f_i(\begin{matrix} x_1, \dots, x_n \end{matrix}), \\ x_{i+1}, \dots, x_n \end{matrix}$$

für ein gewisses $1 \leq i \leq n$

Beobachtungen



$$f_1 = f_1(x_1, \dots, x_n)$$

f_1, \dots, f_n wenden Operationen auf x_1, \dots, x_n an, die intuitiv der Vereinigung oder dem Schnitt von Wertemengen entsprechen.

Außerdem würden wir gerne Aussagen machen können, wie zuverlässig wir einen Fixpunkt finden können.

Verbände



Menge L



Geordnete Menge L, \sqsubseteq

Binäre Relation \sqsubseteq

(reflexiv, antisymmetrisch, transitiv)

Verbände



Verband $V = (L, \subseteq, \cup, \cap, \sqcup)$

Wie geordnete Menge (L, \subseteq)

und außerdem

Jede Untermenge $X \subseteq L$ hat eine
kleinste obere Schranke

Verbände



Weitere Bemerkungen

1. \inf nennt man auch Infimum
2. \sup nennt man auch Supremum
3. Jede Untermenge des Verbandes hat eine größte untere Schranke

Idee



$$_0 = \{ a \mapsto \quad, b \mapsto \quad \}$$

$$_1 = \{ a \mapsto 5, b \mapsto \quad \}$$

$$_2 = \{ a \mapsto 5, b \mapsto 2 \}$$

$$_3 = \{ a \mapsto 7, b \mapsto 2 \}$$

Idee

A horizontal yellow brushstroke with a textured, painterly appearance, extending across the width of the slide below the title.

Wähle zur Darstellung der Werte der Variablen eine verbandartige Datenstruktur

Idee



Wähle zur Darstellung der Werte der Variablen eine verbandartige Datenstruktur

Dann sind x_0, \dots, x_n wieder verbandartig!

Intervallverbände

Intervalle als Wertemengen

Intervalle von Ganzzahlen $I(\mathbb{Z})$

sei das leere Intervall

\top sei $[-, +]$

$A = \mathbb{Z} \quad \{-\}, B = \mathbb{Z} \quad \{+\}$

$I(\mathbb{Z}) = \{ \} \quad \{ [l, u] \mid l \in A \wedge u \in B \wedge l < u \}$

$[l_1, u_1] \subseteq [l_2, u_2] \quad l_1 \leq l_2 \wedge u_1 \leq u_2$

Intervallverbände

$$[l_1, u_1] \sqcup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)]$$

$$[l_1, u_1] \sqcap [l_2, u_2] =$$

falls
 $\max(l_1, l_2) >$
 $\min(u_1, u_2)$

$$[\max(l_1, l_2), \min(u_1, u_2)]$$

sonst

Kombination zweier Verbände

$$L_1 = D_1, \subseteq_1, \quad 1, \top_1, \sqcap_1, \sqcup_1$$

$$L_2 = D_2, \subseteq_2, \quad 2, \top_2, \sqcap_2, \sqcup_2$$

$$L_{1 \times 2} = D_1 \times D_2, \subseteq, \quad , \top, \sqcap, \sqcup$$

$$l_1, l_2 \subseteq l_1', l_2' \quad l_1 \subseteq_1 l_1' \quad l_2 \subseteq_2 l_2'$$

$$l_1, l_2 \sqcup l_1', l_2' = l_1 \sqcup_1 l_1', l_2 \sqcup_2 l_2'$$

$$l_1, l_2 \sqcap l_1', l_2' = l_1 \sqcap_1 l_1', l_2 \sqcap_2 l_2'$$

Kombination zweier Verbände

$$L_{1 \times 2} = D_1 \times D_2, \subseteq, \supseteq, \top, \perp, \sqcup$$

$$l_1, l_2 \subseteq l_1', l_2' \quad l_1 \subseteq_1 l_1' \quad l_2 \subseteq_2 l_2'$$

$$l_1, l_2 \sqcup l_1', l_2' = l_1 \sqcup_1 l_1', l_2 \sqcup_2 l_2'$$

$$l_1, l_2 \sqcap l_1', l_2' = l_1 \sqcap_1 l_1', l_2 \sqcap_2 l_2'$$

$$\top = \top_1, \top_2$$

$$= 1, 2$$

Umgebungsverbände

Da $I(Z), \sqsubseteq, \top$ ein Verband ist, folgt

Jede Umgebung
 $: \{ a_1 \mapsto v_1, \dots, a_n \mapsto v_n \}$
 ist ebenfalls ein Verband

$$I_1, \dots, I_n \sqsubseteq I_1', \dots, I_n' \quad I_1 \sqsubseteq_1 I_1' \quad \dots \quad I_n \sqsubseteq_n I_n'$$

$$\top = [-, +], \dots, [-, +]$$

Verbände von



Analog:

$P = \{a_1, \dots, a_n\}$ ist ein Verband
mit einer Ordnung \sqsubseteq !

Mit \sqsubseteq als Ordnung können
wir nun $\text{lfp}(f_P)$ bestimmen

Monotonie



Funktion F L L ist monoton steigend

$$x, y \in L : x \sqsubseteq y \implies F(x) \sqsubseteq F(y)$$

Beobachtungen



$$x_i = f_i(x_1, \dots, x_n)$$

f_1, \dots, f_n wenden Operationen auf x_1, \dots, x_n an, die intuitiv der Vereinigung oder dem Schnitt von Wertemengen entsprechen.

Außerdem würden wir gerne Aussagen machen können, wie zuverlässig wir einen Fixpunkt finden können.

Beobachtungen



$$i = f_i(x_1, \dots, x_n)$$

$$i = m(i,1) \sqcup \dots \sqcup m(i,ki) \text{ (lfp)}$$

oder

$$i = m(i,1) \sqcap \dots \sqcap m(i,ki) \text{ (gfp)}$$

Außerdem würden wir gerne Aussagen machen können, wie zuverlässig wir einen Fixpunkt finden können.

Beobachtungen

$$i = f_i(x_1, \dots, x_n)$$

$$i = m(i,1) \sqcup \dots \sqcup m(i,ki) \text{ (lfp)}$$

oder

$$i = m(i,1) \sqcap \dots \sqcap m(i,ki) \text{ (gfp)}$$

Falls alle f_i monoton sind, finden wir nach endlich vielen Schritten einen Fixpunkt (aufsteigende Kettenbedingung).

gfp vs. lfp

Größter Fixpunkt

Beginne mit $a_i = \top$ für alle i

$$f_i(x_1, \dots, x_n) = m(i,1) \sqcap \dots \sqcap m(i,k_i)$$

alle f_i sind monoton fallend

Kleinster Fixpunkt

Beginne mit $a_i = \perp$ für alle i

$$f_i(x_1, \dots, x_n) = m(i,1) \sqcup \dots \sqcup m(i,k_i)$$

alle f_i sind monoton steigend

Andere Verbände



■ Bitfelder

- beliebig exakt
- Clustering der Wertemenge denkbar
- ineffizient für große Wertemengen

■ Mehrfachintervalle

- beliebig exakt
- ungünstig für Randfälle
- schwierig zu formalisieren

Probleme



- Unschärfe der abstrakten Wertemenge
- Berechnungsaufwand
- Aliasing
 - Zeiger, Referenzen, Äquivalenzen
 - Parameter in Funktionsaufrufen
- Effekte wie z.B. Timing in Multithread-Systemen schwer abstrakt vorhersagbar

Noch Fragen?

