

# Korrekte Software: Grundlagen und Methoden Vorlesung 9 vom 23.05.16: Weitere Datentypen: Strukturen und Felder

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016



## Fahrplan

- ▶ Einführung
- ▶ Die Floyd-Hoare-Logik
- ▶ Operationale Semantik
- ▶ Denotationale Semantik
- ▶ Äquivalenz der Semantiken
- ▶ Verifikation: Vorwärts oder Rückwärts?
- ▶ Korrektheit des Hoare-Kalküls
- ▶ Einführung in Isabelle/HOL
- ▶ **Weitere Datentypen: Strukturen und Felder**
- ▶ Funktionen und Prozeduren
- ▶ Referenzen und Zeiger
- ▶ Frame Conditions & Modification Clauses
- ▶ Ausblick und Rückblick



## Motivation

- ▶ Weitere Basisdatentypen von C (arrays, strings und structs)
- ▶ Noch rein funktional, keine Pointer



## Arrays

```
int a[1][2];

bool b[][] = { {1, 0},
               {1, 1},
               {0, 0} }; /* Ergibt Array [3][2] */

printf(b[2][1]); /* liefert '0' */

int six[6] = {1,2,3,4,5,6};

// Allgemeine Form

typ name[groesse1][groesse2]...[groesseN] =
{ ... }
x;
```



## Strings

```
char hallo[5] = {'h', 'a', 'l', 'l', 'o', '\0'};
char hallo[] = "hallo";
printf(hallo[4]); /* liefert 'o' */
```



## Struct

```
struct Vorlesung {
    char dozenten[2][30];
    char titel[30];
    int cp;
} ksgm;

struct Vorlesung ksgm;

int i = 0;
char name1[] = "Serge Autexier";
while (i < strlen(name1)) {
    ksgm.dozenten[0][i] = name1[i];
    i = i + 1;
}
char name2[] = "Christoph Lueth";
i = 0;
while (i < strlen(name2)) {
    ksgm.dozenten[1][i] = name2[i];
    i = i + 1;
}
ksgm.cp = 6;
```



## Rekursive Struct

```
struct Liste {
    int kopf;
    Liste *rest;
} start;

start.kopf = 10; /* start.rest bleibt undefiniert */

int i = 9;
while (i > 0) {
    struct Liste next;
    next.kopf = i;
    next.rest = start;
    i = i - 1;
    start = next;
}
```



## Ausdrücke

Location Expressions **Lexp** ::= **Loc** | **Lexp** [a] | **Lexp** . name

**Aexp** a ::= **N** | **Lexp** | a<sub>1</sub>+a<sub>2</sub> | a<sub>1</sub>-a<sub>2</sub> | a<sub>1</sub>\*a<sub>2</sub> | a<sub>1</sub>/a<sub>2</sub> | strlen(Exp)

**Bexp** b ::= **0** | **1** | a<sub>1</sub>== a<sub>2</sub> | a<sub>1</sub>!= a<sub>2</sub> | a<sub>1</sub><= a<sub>2</sub> | !b | b<sub>1</sub>&& b<sub>2</sub> | b<sub>1</sub>|| b<sub>2</sub>

**Exp** e ::= **Aexp** | **Bexp** | **C**

**ExpList** el ::= e (, el)?



## Statements

Type  $type ::= int \mid char \mid struct \text{ name } \{puredecl^*\}$   
 Decl  $decl ::= puredecl$   
           |  $type \text{ Loc}[] = \{el\};$   
 puredecl  $::= type \text{ Loc};$   
           |  $type \text{ Loc}[N];$   
 Stmt  $c ::= decl$   
           | **Lexp** = **Exp**;  
           | **if** (  $b$  )  $c_1$  **else**  $c_2$   
           | **while** (  $b$  )  $c$   
           |  $\{c^*\}$



## Werte und Zustände

Container **Cont** ::= **Loc** | **Cont** [N] | **Cont** . name

Werte sind die kleinste Menge **V** für die gilt

► **N**, **B**, **C** sind Teilmengen von **V** (V<sub>B</sub>)

Zustände sind partielle Funktionen  $\sigma : \mathbf{Cont} \rightarrow \mathbf{V}$  so dass gilt

►  $\forall c, c' \in \text{Dom}(\sigma). c$  ist kein Präfix von  $c'$  und umgekehrt.

► if  $c[i]c' \in \text{Dom}(\sigma)$  then  $\forall 0 \leq j \leq i. \exists c_j. c[j]c_j \in \text{Dom}(\sigma)$

Zustandprojektion Sei  $u \in \mathbf{Cont}$  und  $\sigma$  ein Zustand: Wir definieren die Projektion von  $\sigma$  auf  $u$  durch

$$\sigma|_u := \{(v, n) \mid (uv, n) \in \sigma\}$$



## Beispiel

Programm

Zustand

```
struct A {
  int c[2];
  struct B {
    char name[20];
  } b;
}
struct A x[] = {
  {{1,2},
  {{ 'n', 'a', 'm', 'e', '1', '\0' }}},
  {{3,4},
  {{ 'n', 'a', 'm', 'e', '2', '\0' }}},
}
```

$x.[0].c[0] \rightarrow 1$	$x.[1].c[0] \rightarrow 3$
$x.[0].c[1] \rightarrow 2$	$x.[1].c[1] \rightarrow 4$
$x.[0].b.name[0] \rightarrow 'n'$	$x.[1].b.name[0] \rightarrow 'n'$
$x.[0].b.name[1] \rightarrow 'a'$	$x.[1].b.name[1] \rightarrow 'a'$
$x.[0].b.name[2] \rightarrow 'm'$	$x.[1].b.name[2] \rightarrow 'm'$
$x.[0].b.name[3] \rightarrow 'e'$	$x.[1].b.name[3] \rightarrow 'e'$
$x.[0].b.name[4] \rightarrow '1'$	$x.[1].b.name[4] \rightarrow '2'$
$x.[0].b.name[5] \rightarrow '\0'$	$x.[1].b.name[5] \rightarrow '\0'$



## Auswertung von Lexp zu Cont

$$\frac{x \in \mathbf{Loc}}{\langle x, \sigma \rangle \rightarrow_{\text{Lexp}} x}$$

$$\frac{\langle \text{lexp}, \sigma \rangle \rightarrow_{\text{Lexp}} c \quad \langle a, \sigma \rangle \rightarrow_{\text{Aexp}} i}{\langle \text{lexp}[a], \sigma \rangle \rightarrow_{\text{Lexp}} c[i]}$$

$$\frac{\langle \text{lexp}, \sigma \rangle \rightarrow_{\text{Lexp}} c}{\langle \text{lexp.name}, \sigma \rangle \rightarrow_{\text{Lexp}} c.name}$$



## Aexp: Operationale Semantik

$$\frac{\langle \text{lexp}, \sigma \rangle \rightarrow_{\text{Lexp}} c \quad c \in \text{Dom}(\sigma)}{\langle \text{lexp}, \sigma \rangle \rightarrow_{\text{Aexp}} \sigma(c)}$$

$$\frac{\langle \text{lexp}, \sigma \rangle \rightarrow_{\text{Lexp}} c \quad c \notin \text{Dom}(\sigma)}{\langle \text{lexp}, \sigma \rangle \rightarrow_{\text{Aexp}} \perp}$$

$$\frac{\langle \text{str}, \sigma \rangle \rightarrow_{\text{Lexp}} s :: \text{char}[n], \quad l = \min(\{n+1\} \cup \{m \mid m < n, s[m] = '\0', s[0..m-1] \neq '\0'\})}{\langle \text{strlen}(\text{str}), \sigma \rangle \rightarrow_{\text{Aexp}} l}$$



## Operationale Semantik: Zuweisungen

$$\frac{\langle \text{lexp}, \sigma \rangle \rightarrow_{\text{Lexp}} c \quad \sigma(c) :: \tau \quad \langle \text{exp}, \sigma \rangle \rightarrow e :: \tau}{\langle \text{lexp} = \text{exp}, \sigma \rangle \rightarrow_{\text{Stmt}} \sigma[e/c]}$$

Stmt  $c ::= decl$   
           | **Lexp** = **Exp**;  
           | **if** (  $b$  )  $c_1$  **else**  $c_2$   
           | **while** (  $b$  )  $c$   
           |  $\{c^*\}$



## Denotationale Semantik

► Denotation für **Lexp**

$$\mathcal{L}[\mathbf{x}] = \{(\sigma, x) \mid \sigma \in \Sigma\}$$

$$\mathcal{L}[\text{lexp}[a]] = \{(\sigma, l[i]) \mid (\sigma, l) \in \mathcal{L}[\text{lexp}], (\sigma, i) \in \mathcal{E}[a]\}$$

$$\mathcal{L}[\text{lexp.name}] = \{(\sigma, l.name) \mid (\sigma, l) \in \mathcal{L}[\text{lexp}]\}$$

► Denotation für **Zuweisungen**

$$\mathcal{D}[\text{lexp} = \text{exp}] = \{(\sigma, \sigma[e/c]) \mid (\sigma, c) \in \mathcal{L}[\text{lexp}], (\sigma, e) \in \mathcal{E}[\text{exp}]\}$$



## Hoare-Regel

► Vor- Nachbedingungen von Hoare-Regeln müssen auch Gleichungen über Container Werte haben

► Nicht unbedingt alle, aber alle die gebraucht werden



## Beispiel

```
int a[3];
/** { 1 } */
/** { 3 = 3 and 3 = 3 } */
a[2] = 3;
/** { a[2] = 3 and a[2] = 3 } */
/** { 4 = 4 and a[2] = 3 and 4 * a[2] = 12 } */
a[1] = 4;
/** { a[1] = 4 and a[2] = 3 and a[1] * a[2] = 12 } */
/** { 5 = 5 and a[1] = 4 and a[2] = 3 and
    5 * a[1] * a[2] = 60 } */
a[0] = 5;
/** { a[0] = 5 and a[1] = 4 and a[2] = 3 and
    a[0] * a[1] * a[2] = 60 } */
```



## Beispiel

```
int a[3];
/** { true } */
/** { 2 = 2 and 3 = 3 and 3 = 3 } */
int i = 2;
/** { i = 2 and 3 = 3 and 3 = 3 } */
a[i] = 3;
/** { i = 2 and a[i] = 3 and a[i] = 3 } */
/** { 1 = 1 and 4 = 4 and a[2] = 3 and 4 * a[2] = 12 } */
i = 1;
/** { i = 1 and 4 = 4 and a[2] = 3 and 4 * a[2] = 12 } */
a[i] = 4;
/** { i = 1 and a[i] = 4 and a[2] = 3 and
    a[i] * a[2] = 12 } */
/** { 0 = 0 and a[1] = 4 and a[2] = 3 and
    a[1] * a[2] = 12 } */
i = 0;
/** { i = 0 and a[1] = 4 and a[2] = 3 and
    a[1] * a[2] = 12 } */
/** { i = 0 and 5 = 5 and a[1] = 4 and a[2] = 3 and
    5 * a[1] * a[2] = 60 } */
a[i] = 5;
/** { i = 0 and a[i] = 5 and a[1] = 4 and a[2] = 3 and
    a[i] * a[1] * a[2] = 60 } */
/** { i = 0 and a[i] = 5 and a[1] = 4 and a[2] = 3 and
    a[0] * a[1] * a[2] = 60 } */
```

