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;
}
```

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 $\textbf{Lexp} ::= \textbf{Loc} \mid \textbf{Lexp}\ [a] \mid \textbf{Lexp}\ .\ \textbf{name}$

Aexp $a ::= \textbf{N} \mid \textbf{Lexp} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 / a_2 \mid strlen(Exp)$

Bexp $b ::= \textbf{0} \mid \textbf{1} \mid a_1 == a_2 \mid a_1 != a_2$
$\mid a_1 <= a_2 \mid !b \mid b_1 \mathbin{\&\&} b2 \mid b_1 \mathbin{||} b_2$

Exp $e := \textbf{Aexp} \mid \textbf{Bexp} \mid \textbf{C}$

ExpList $el := e\ (, el)?$

Statements

$$\text{Type} \quad \text{type} ::= \mathit{int} \mid \mathit{char} \mid \mathit{struct}\ \textbf{name}\ \{\mathit{puredecl}*\}$$

$$\begin{array}{rrcl} \text{Decl} & \mathit{decl} & ::= & \mathit{puredecl} \\ & & \mid & \mathit{type}\ \textbf{Loc}[\,]\ =\ \{\mathit{el}\}; \end{array}$$

$$\begin{array}{rcl} \text{puredecl} & ::= & \mathit{type}\ \textbf{Loc}; \\ & \mid & \mathit{type}\ \textbf{Loc}[\textbf{N}]; \end{array}$$

$$\begin{array}{rrcl} \text{Stmt} & c & ::= & \text{decl} \\ & & \mid & \textbf{Lexp} = \textbf{Exp}; \\ & & \mid & \textbf{if}\ (\ b\ )\ c_1\ \ \textbf{else}\ \ c_2 \\ & & \mid & \textbf{while}\ (\ b\ )\ c \\ & & \mid & \{c^*\} \end{array}$$

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_B)

Zustände sind partielle Funktionen σ : **Cont** $\rightharpoonup$ **V** so dass gilt

- $\forall c, c' \in Dom(\sigma).c$ ist kein Präfix von c' und umgekehrt.
- if $c[i]c' \in Dom(\sigma)$ then $\forall 0 \leq j \leq i.\exists c_j.c[j]c_j \in Dom(\sigma)$

Zustandprojektion Sei $u \in$ **Cont** und σ ein Zustand: Wir definieren die Projektion von σ auf u durch

$$\sigma_{|u} := \{(v, n) | (uv, n) \in \sigma\}$$

Beispiel

Programm

```
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'}}
 }
};
```

Zustand

$x.[0].c[0] \rightarrow 1$

$x.[0].c[1] \rightarrow 2$

$x.[0].b.name[0] \rightarrow `n'$

$x.[0].b.name[1] \rightarrow `a'$

$x.[0].b.name[2] \rightarrow `m'$

$x.[0].b.name[3] \rightarrow `e'$

$x.[0].b.name[4] \rightarrow `1'$

$x.[0].b.name[5] \rightarrow `\backslash 0'$

$x.[1].c[0] \rightarrow 3$

$x.[1].c[1] \rightarrow 4$

$x.[1].b.name[0] \rightarrow `n'$

$x.[1].b.name[1] \rightarrow `a'$

$x.[1].b.name[2] \rightarrow `m'$

$x.[1].b.name[3] \rightarrow `e'$

$x.[1].b.name[4] \rightarrow `2'$

$x.[1].b.name[5] \rightarrow `\backslash 0'$

Auswertung von Lexp zu Cont

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

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

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

Aexp: Operationale Semantik

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

$$\frac{\langle lexp, \sigma \rangle \rightarrow_{Lexp} c \qquad c \notin Dom(\sigma)}{\langle lexp, \sigma \rangle \rightarrow_{Aexp} \bot}$$

$$\frac{\begin{array}{c}\langle str, \sigma \rangle \rightarrow_{Lexp} s :: char[n], \\ l = min(\{n+1\} \cup \{m | m < n, s[m] =' \backslash 0', s[0..m-1] \neq' \backslash 0'\}\end{array}}{\langle strlen(str), \sigma \rangle \rightarrow_{Aexp} l}$$

Operationale Semantic: Zuweisungen

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

$$\begin{array}{lll} \text{Stmt } c & ::= & \text{decl} \\ & | & \textbf{Lexp} = \textbf{Exp}; \\ & | & \textbf{if} \ (\ b\ )\ c_1\ \ \textbf{else}\ \ c_2 \\ & | & \textbf{while}\ (\ b\ )\ c \\ & | & \{c^*\} \end{array}$$

Denotationale Semantik

- Denotation für **Lexp**

$$\mathcal{L}[\![x]\!] = \{(\sigma, x) | \sigma \in \Sigma\}$$
$$\mathcal{L}[\![lexp[a]]\!] = \{(\sigma, l[i]) | (\sigma, l) \in \mathcal{L}[\![lexp]\!], (\sigma, i) \in \mathcal{E}[\![a]\!]\}$$
$$\mathcal{L}[\![lexp.name]\!] = \{(\sigma, l.name) | (\sigma, l) \in \mathcal{L}[\![lexp]\!]\}$$

- Denotation für **Zuweisungen**

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

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 } */
```