Korrekte Software: Grundlagen und Methoden
Vorlesung 12 vom 09.06.16: Referenzen und Speichermodelle

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016

Organisatorisches

Die Vorlesung am Montag, 13.06.2016 fällt wegen der 10-Jahres-Feier des DFKI Bremen aus.

Besucht unseren Tag der offenen Tür am Dienstag, 14.06.2016 (Robert-Hooke-Straße 1, hinter dem Fallturm).

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

- Bisher: Zustand ist **Loc** $\rightharpoonup$ **Val**
  - **Loc** — symbolische Zustände (*Locations*)
  - **Val** — Basisdatentypen
- Grenzen: keine Referenzen
  - Damit auch kein *call by reference*
  - Funktion können nur globale Seiteneffekte haben
  - Was wäre C ohne Pointer?

Refenzen in C

- Pointer in C ("pointer type"):
  - Schwach getypt (**void** $*$ kompatibel mit allen Zeigertypen)
  - Eingeschränkte Zeigerarithmetik (Addition, Subtraktion)
  - Felder werden durch Zeigerarithmetik implementiert
- Pointer sind *first-class-values*
- C-Standard läßt das Speichermodell relativ offen
  - Repräsentation von Objekten

Erweiterung des Zustandmodells

- Erweiterung von Zustand und Werten:

$$\Sigma = \mathbf{Loc} \rightharpoonup \mathbf{Val} \qquad \mathbf{Val} = \mathbf{N} + \mathbf{C} + \mathbf{Loc}$$

- Was ist **Loc**?
  - Locations (Speicheradressen)
  - Man kann **Loc** axiomatisch oder modellbasiert beschreiben.

Axiomatisches Zustandsmodell

- Der Zustand ist ein abstrakter Datentyp Σ mit zwei Operationen und folgenden Gleichungen:

$$\mathit{read} : \Sigma \to \mathbf{Loc} \rightharpoonup \mathbf{Val}$$
$$\mathit{upd} : \Sigma \to \mathbf{Loc} \to \mathbf{Val} \rightharpoonup \Sigma$$

$$\begin{aligned}
\mathit{read}(\mathit{upd}(\sigma, l, v), l) &= v \\
l \neq m \longrightarrow \mathit{read}(\mathit{upd}(\sigma, l, v), m) &= \mathit{read}(\sigma, m) \\
\mathit{upd}(\mathit{upd}(\sigma, l, v), l, w) &= \mathit{upd}(\sigma, l, w) \\
l \neq m \longrightarrow \mathit{upd}(\mathit{upd}(\sigma, l, v), m, w) &= \mathit{upd}(\mathit{upd}(\sigma, m, w), l, v)
\end{aligned}$$

- Diese Gleichungen sind vollständig.

Axiomatisches Speichermodell

- Es gibt einen leeren Speicher, und neue ("frische") Adressen:

$$\begin{aligned} \mathit{empty} &: \Sigma \\ \mathit{fresh} &: \Sigma \to \mathbf{Loc} \\ \mathit{rem} &: \Sigma \to \mathbf{Loc} \to \Sigma \end{aligned}$$

  - *fresh* modelliert Allokation, *rem* modelliert Deallokation
- *dom* beschreibt den Definitionsbereich:

$$\begin{aligned} \mathit{dom}(\sigma) &= \{l \mid \exists v.\, \mathit{read}(\sigma, l) = v\} \\ \mathit{dom}(\mathit{empty}) &= \emptyset \end{aligned}$$

- Eigenschaften von *empty*, *fresh* und *rem*:

$$\begin{aligned} \mathit{fresh}(\sigma) &\notin \mathit{dom}(\sigma) \\ \mathit{dom}(\mathit{rem}(\sigma, l)) &= \mathit{dom}(\sigma) \setminus \{l\} \\ l \neq m \longrightarrow \mathit{read}(\mathit{rem}(\sigma, l), m) &= \mathit{read}(\sigma, m) \end{aligned}$$

Zeigerarithmetik

- Erklärt noch keine Zeigerarithmetik — dazu:

$$add : \mathbf{Loc} \to \mathbb{Z} \to \mathbf{Loc}$$

  - Wir betrachten keine Differenz von Zeigern

$$add(l, 0) = l$$
$$add(add(l, a), b) = add(l, a + b)$$

Erweiterung der Semantik

- Problem: **Loc** haben unterschiedliche Semantik auf der linken oder rechten Seite einer Zuweisung.
  - x = x+1 — Links: Addresse der Variablen, rechs: Wert an dieser Adresse
- Lösung: “Except when it is (. . .) the operand of the unary & oprerator, the left operand of the . operator or an assigment operator, an lvalue that does not have array type is converted to the value stored in the designated object (and is no longer an lvalue)” *C99 Standard*, §6.3.2.1 (2)

Erweiterung der Semantik: Lexp

$$\mathcal{L}[\![-]\!] : \mathit{Env} \to \mathbf{Lexp} \to \Sigma \rightharpoonup \mathbf{Loc}$$

$$\begin{aligned}
\mathcal{L}[\![x]\!]\ \Gamma &= \{(\sigma, \Gamma!x) \mid \sigma \in \Sigma\} \\
\mathcal{L}[\![\mathit{lexp}[a]]\!]\ \Gamma &= \{(\sigma, \mathit{add}(l, i \cdot \mathit{sizeof}(\tau))) \mid (\sigma, l) \in \mathcal{L}[\![\mathit{lexp}]\!]\ \Gamma, (\sigma, i) \in \mathcal{E}[\![a]\!]\ \Gamma\} \\
&\qquad \mathit{type}(\Gamma, \mathit{lexp}) = \tau \text{ ist der Basistyp des Feldes} \\
\mathcal{L}[\![\mathit{lexp}.f]\!]\ \Gamma &= \{(\sigma, l.f) \mid (\sigma, \mathit{add}(l, \mathit{fld_off}(\tau, f))) \in \mathcal{L}[\![\mathit{lexp}]\!]\ \Gamma\} \\
&\qquad \mathit{type}(\Gamma, \mathit{lexp}) = \tau \text{ ist der Typ der Struktur} \\
\mathcal{L}[\![*e]\!]\ \Gamma &= \mathcal{E}[\![e]\!]\ \Gamma
\end{aligned}$$

- $\mathit{type}(\Gamma, e)$ ist der Typ eines Ausdrucks
- $\mathit{fld\_off}(\tau, f)$ ist der Offset des Feldes f in der Struktur τ
- $\mathit{sizeof}(\tau)$ ist die Größe von Objekten des Typs τ

Erweiterung der Semantik: Aexp(1)

$$\mathcal{E}[\![-]\!] : \mathit{Env} \to \mathbf{Aexp} \to \Sigma \rightharpoonup \mathbf{Val}$$

$$\begin{aligned}
\mathcal{E}[\![n]\!]\ \Gamma &= \{(\sigma, n) \mid \sigma \in \Sigma\} \quad \text{für } n \in \mathbf{N} \\
\mathcal{E}[\![e]\!]\ \Gamma &= \{(\sigma, \mathit{read}(\sigma, l)) \mid (\sigma, l) \in \mathcal{L}[\![e]\!]\ \Gamma\} \\
&\qquad e \equiv x \mid \mathit{lexp}[a] \mid \mathit{lexp}.n \mid *e,\ \mathit{type}(\Gamma, e) \text{ kein Array-Typ} \\
\mathcal{E}[\![e]\!]\ \Gamma &= \{(\sigma, l) \mid (\sigma, l) \in \mathcal{L}[\![e]\!]\ \Gamma\} \\
&\qquad e \equiv x \mid \mathit{lexp}[a] \mid \mathit{lexp}.n \mid *e,\ \mathit{type}(\Gamma, e) \text{ Array-Typ} \\
\mathcal{E}[\![\&e]\!]\ \Gamma &= \{(\sigma, l) \mid (\sigma, l) \in \mathcal{L}[\![e]\!]\ \Gamma\} \\
\mathcal{E}[\![p + e]\!]\ \Gamma &= \{(\sigma, \mathit{add}(l, n \cdot \mathit{sizeof}(\tau))) \mid (\sigma, l) \in \mathcal{L}[\![p]\!]\ \Gamma \wedge (\sigma, n) \in \mathcal{E}[\![e]\!]\ \Gamma\} \\
&\qquad \mathit{type}(\Gamma, p) = *\tau,\ \mathit{type}(\Gamma, a_1) \text{ Integer-Typ} \\
\mathcal{E}[\![e + p]\!]\ \Gamma &= \{(\sigma, \mathit{add}(l, n \cdot \mathit{sizeof}(\tau))) \mid (\sigma, n) \in \mathcal{E}[\![e]\!]\ \Gamma \wedge (\sigma, l) \in \mathcal{L}[\![p]\!]\ \Gamma\} \\
&\qquad \mathit{type}(\Gamma, e) \text{ Integer-Typ und } \mathit{type}(\Gamma, p) = *\tau
\end{aligned}$$

Erweiterung der Semantik: Aexp(2)

$$\mathcal{E}[\![-]\!] : \mathit{Env} \to \mathbf{Aexp} \to \Sigma \rightharpoonup \mathbf{Val}$$

$$\mathcal{E}[\![a_0 + a_1]\!]\ \Gamma = \{(\sigma, n_0 + n_1 \mid (\sigma, n_0) \in \mathcal{E}[\![a_0]\!]\ \Gamma \wedge (\sigma, n_1) \in \mathcal{E}[\![a_1]\!]\ \Gamma\}$$

für a_0, a_1 arithmetische Typen

$$\mathcal{E}[\![a_0 - a_1]\!]\ \Gamma = \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in \mathcal{E}[\![a_0]\!]\ \Gamma \wedge (\sigma, n_1) \in \mathcal{E}[\![a_1]\!]\ \Gamma\}$$

$$\mathcal{E}[\![a_0 * a_1]\!]\ \Gamma = \{(\sigma, n_0 * n_1) \mid (\sigma, n_0) \in \mathcal{E}[\![a_0]\!]\ \Gamma \wedge (\sigma, n_1) \in \mathcal{E}[\![a_1]\!]\ \Gamma\}$$

$$\mathcal{E}[\![a_0/a_1]\!]\ \Gamma = \{(\sigma, n_0/n_1) \mid (\sigma, n_0) \in \mathcal{E}[\![a_0]\!]\ \Gamma \wedge (\sigma, n_1) \in \mathcal{E}[\![a_1]\!]\ \Gamma \wedge n_1 \neq 0\}$$

Übersicht: Typen in C

int, **char**	Integer-Typ	arithmet. Typen	
float, **double**	Fließkomma-Typ		skalare Typen
* t	Pointer-Typ		
t[i]	Array-Typ		
struct t {...}	Struktur-Typen		
struct t, t[]	unvollständige Typen		

Hoare-Triple

$$\models \{P\}\, c\, \{Q|R\}$$

- $P, Q, R : \Sigma \to \mathit{Bool}$ explizite Zustandsprädikate
- Übersetzung $[\![.]\!]$ von logischen Formeln in Zustandsprädikate
- Beispiel:

$$[\![x > 0]\!]\ \Gamma = \lambda\sigma.\, \mathit{read}(\sigma, \Gamma!x) > 0$$

- Für kürzere Regeln: "Lifting" von Boolschen Operationen:

$$P \wedge Q \stackrel{def}{=} \lambda\sigma.\, P(\sigma) \wedge Q(\sigma)$$

$$\neg P \stackrel{def}{=} \lambda\sigma.\, \neg P(\sigma)$$

$$P \longrightarrow Q \stackrel{def}{=} \lambda\sigma.\, P(\sigma) \longrightarrow Q(\sigma)$$

Regeln des Hoare-Kalküls

$$\frac{}{\Gamma \vdash \{\lambda\sigma.Q(upd(\sigma, [\![l]\!]\Gamma, [\![e]\!]\Gamma))\}\, l = e\, \{Q|R\}}$$

$$\frac{}{\Gamma \vdash \{P\}\,\{\}\,\{P|R\}} \qquad \frac{\Gamma \vdash \{P\}\, c\, \{Q_1|R\} \qquad \Gamma \vdash \{Q_1\}\,\{c_s\}\,\{Q_2|R\}}{\Gamma \vdash \{P\}\,\{c\ c_s\}\,\{Q_2|R\}}$$

$$\frac{\Gamma \vdash \{\ P \wedge [\![b]\!]\Gamma\ \}\, c_0\, \{Q|R\} \qquad \Gamma \vdash \{\ P \wedge \neg[\![b]\!]\Gamma\ \}\, c_1\, \{Q|R\}}{\Gamma \vdash \{P\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{Q|R\}}$$

$$\frac{\Gamma \vdash \{\ P \wedge [\![b]\!]\Gamma\ \}\, c\, \{Q|R\}}{\Gamma \vdash \{P\}\ \textbf{while}(b)\ c\, \{\ Q \wedge \neg[\![b]\!]\Gamma\ |R\}}$$

$$\frac{P' \longrightarrow P \qquad \Gamma \vdash \{P\}\, c\, \{Q|R\} \qquad Q \longrightarrow Q'}{\Gamma \vdash \{P'\}\, c\, \{Q'|R\}}$$

Ein kurzes Beispiel

```
void foo(){
int x, y, *z; /* Locations: l, m, n */

/** \s. read(upd(upd(upd(s, n, l), l, 0),
          read(upd(upd(s, n, l), l, 0), n), 5), l) = 5 */
z= &x;
/** \s. read(upd(upd(s, l, 0),
                 read(upd(s, l, 0), n), 5), l) = 5 */ (3)
x= 0;
/** \s. read(upd(s, read(s, n), 5), l) = 5) */ (2)
*z= 5;
/** \s. read(s, l) = 5 */                          (1)
/** \s. read(upd(s, m, read(s, l)), m) = 5 */
y= x;
/** \s. read(s, m) = 5
/** { y == 5 } */
```

Ein kurzes Beispiel

- An der Stelle (1) können wir direkt vereinfachen
- An den Stellen (2) und (3) ist keine Zwischenvereinfachung mehr möglich
- Die finale Vorbedingung wird wie folgt vereinfacht:

$$read(upd(upd(upd(\sigma, n, I), I, 0), \underline{read(upd(}upd(\sigma, n, I), \underline{I, 0), n)}, 5), I) = 5$$

$$read(upd(upd(upd(\sigma, n, I), I, 0)\underline{read(upd(\sigma, n, I), n)}, 5), I) = 5$$

$$\underline{read(upd(}upd(upd(\sigma, n, I), I, 0), \underline{I, 5), I)} = 5$$

$$5 = 5$$

Zusammenfassung

- Um Referenzen (Pointer) in C behandeln zu können, benötigen wir ein Zustandsmodell
- Referenzen werden zu Werten wie Zahlen oder Zeichen.
  - Arrays und Strukturen sind keine first-class values.
  - Großes Problem: aliasing
- Erweiterung der Semantik und der Hoare-Tripel nötig:
  - Vor/Nachbedingungen werden zu Zustandsprädikaten.
  - Zuweisung wird zu Zustandsupdate.
  - Problem: Vereinfachung von Zuständen benötigt Gleichheit/Ungleichheit von Referenzen
- Nächsten Donnerstag: Gleichheit und Ungleichheit über Loc, Generierung von Vorbedingungen, Definiertheit