Korrekte Software: Grundlagen und Methoden

Vorlesung 13 vom 12.07.22

Referenzen und Speichermodelle

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2022

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül I
- Der Floyd-Hoare-Kalkül II: Invarianten
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Funktionen und Prozeduren I
- Funktionen und Prozeduren II
- Referenzen und Speichermodelle
- Ausblick und Rückblick

Motivation

- Warum Referenzen?
  - Nötig für *call by reference*
  - Funktionen können sonst nur **globale** Seiteneffekte haben
  - Effizienz
- Kurze Begriffsklärung:
  - Referenzen: getypt, eingeschränkte Arithmetik
  - Zeiger: ungetypt, Zeigerarithmetik

Refenzen in C

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

Referenzen in anderen Sprachen

- Java:
  - (Fast) alles ist eine Referenz
  - Schwach getypt (Subtyping und Typumwandlung)
- Haskell, SML, OCaml:
  - Stark getypt (typsicher)
- Scriptsprachen (Python, Ruby):
  - Ähnlich Java

Ausdrücke

- Neue Operatoren: Addressoperator (&) und Derefenzierung ($*$)

$$\begin{array}{rl}
\textbf{Lexp} & l ::= \textbf{Idt} \mid l[a] \mid l.\textbf{Idt} \mid *a \\
\textbf{Aexp} & a ::= \textbf{Z} \mid \textbf{C} \mid \textbf{Lexp} \mid \&l \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 / a_2 \\
\textbf{Bexp} & b ::= \ldots \\
\textbf{Exp} & e := \textbf{Aexp} \mid \textbf{Bexp} \\
\textbf{Stmt} & c ::= \ldots \\
\textbf{Type} & t ::= \textbf{void} \mid \textbf{char} \mid \textbf{int} \mid *t \mid \textbf{struct}\ \textbf{Idt}^{?}\ \{\textbf{Decl}^{+}\} \mid t\ \textbf{Idt}[a]
\end{array}$$

Das Problem mit Zeigern

- **Aliasing:** Verschiedene Bezeichner (**Lexp**) für die gleiche Lokation $l \in$ **Loc**

```
int a;
int *x;

x= &a;
a= 0;
// {a = 0}
*x= 7;
// {a = 7} (*)
```

- Wert von a ändert sich **ohne dass a erwähnt** wird.
- An der Stelle (*) zwei Bezeichner für die gleiche Lokation: a und $*x$
- Großes Problem für Semantik und Hoare-Kalkül.
- Modellierung der Zuweisung durch Substitution nicht mehr möglich

Erweiterung des Zustandmodells

- Bisheriger Zustand $\Sigma \stackrel{def}{=} \mathbf{Loc} \rightharpoonup \mathbf{V}$ mit
  - **Locations** (siehe Vorlesung 8)
  - Werte: $\mathbf{V} = \mathbb{Z}$
- Ansatz reicht nicht mehr: Werte müssen auch Locations sein:

$$\mathbf{V} \stackrel{def}{=} \mathbb{Z} + \mathbf{Loc}$$

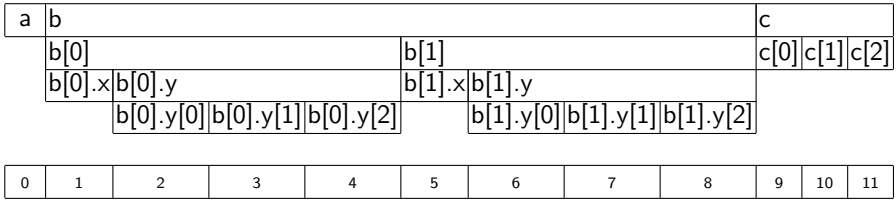
  und somit sind Zustände $\Sigma \stackrel{def}{=} \mathbf{Loc} \rightharpoonup (\mathbb{Z} + \mathbf{Loc})$
- Was ist die Semantik von **Loc**? $\longrightarrow$ Speichermodelle

Speichermodelle I: Compiler

Beispieldeklarationen:

```
int a;
struct {
  int x;
  int y[3]} b[2];
int c[3];
```

Übersetzung in konkretes **Speicherlayout**:



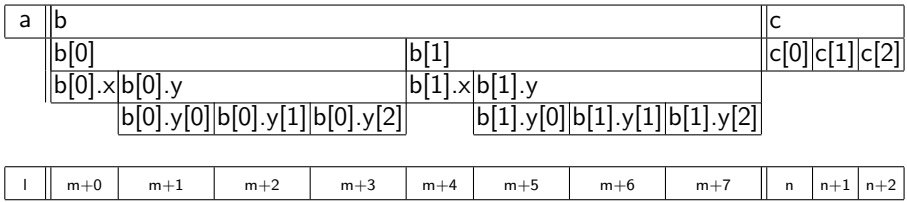
Denotation von $b[0].y[1]$ ist 3

Speichermodelle II: C-Standard

Beispieldeklarationen:

```
int a;
struct {
  int x;
  int y[3]} b[2];
int c[3];
```

Übersetzung in abstraktes **Speicherlayout**:



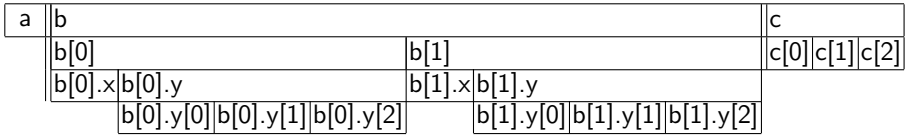
Denotation von $b[0].y[1]$ ist $m + 3$, mit m **unbestimmte** Adresse

Speichermodelle III: Symbolisch

Beispieldeklarationen:

```
int a;
struct {
  int x;
  int y[3]} b[2];
int c[3];
```

Übersetzung in **symbolische Adressen**:



Denotation von $b[0].y[1]$ ist $m[0].y[1]$, mit m unbestimmte Adresse

Speichermodelle im Überblick

- Speichermodell I:
  - Ausführbar, aber **spezifisch** für Compiler und **Architektur**
  - Uneingeschränkte **Zeigerarithmetik**: $\mathbf{Loc} = \mathbb{N}$
- Speichermodell II:
  - Spezifiziert durch den **Standard**
  - **Eingeschränkte** Form der **Zeigerarithmetik** $p + n$ mit p Zeiger, $n \in \mathbb{N}$
- Speichermodell III:
  - **Mischung** von Syntax und Semantik: $a.x$ kann L-Ausdruck, aber auch Location bezeichnen.

Arbeitsblatt 13.1: Jetzt mit Zeigern!

Hier eine weitere Folge von Deklarationen:

```
int *a[1];
struct {
  int p[2];
  struct {
    int x;
    int y; } q[2];
  } b;
```

- Skizziert hier das Speichermodell — konkret, abstrakt, symbolisch.
- Welches sind die jeweiligen Adressen (**Loc**)?
- Was sind die Denotationen für `a[1]`, `b.p[1]`, `b.q[0]`, `b.q[1].y`?
- Welche davon sind definiert/undefiniert?

Arbeitsblatt 13.2: Jetzt mit noch mehr Zeigern!

Hier eine weitere Folge von Deklarationen:

```
int *a[1];
struct {
  int p[2];
  struct {
    int x;
    int y; } *q[2];
  } b;
```

- Skizziert hier das Speichermodell — konkret, abstrakt, symbolisch.
- Welches sind die jeweiligen Adressen (**Loc**)?
- Was sind die Denotationen für `a[1]`, `b.p[1]`, `b.q[0]`, `(*b.q[1]).y`?
- Welche davon sind definiert/undefiniert?

Erweiterung der Semantik

- Problem: **Loc** haben unterschiedliche Semantik auf der linken oder rechten Seite einer Zuweisung.
  - x = x+1 — Links: Addresse der Variablen, rechts: Wert an dieser Adresse
  - Einführung von Derefenzierung und Adressoperator verschärft dieses Problem
- Lösung in C: "Except when it is (...) the operand of the unary & operator, 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)
- Nicht spezifisch für C

Erweiterung der Semantik: Lexp

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

$$\begin{aligned}
[\![x]\!]^{\Gamma}_{\mathcal{L}} &= \{(\sigma, \Gamma(x)) \mid \sigma \in \Sigma\} \\
[\![m[a]]\!]^{\Gamma}_{\mathcal{L}} &= \{(\sigma, l[i]) \mid (\sigma, l) \in [\![m]\!]^{\Gamma}_{\mathcal{L}}, (\sigma, i) \in [\![a]\!]^{\Gamma}_{\mathcal{A}}\} \\
[\![m.f]\!]^{\Gamma}_{\mathcal{L}} &= \{(\sigma, l.f) \mid (\sigma, l) \in [\![m]\!]^{\Gamma}_{\mathcal{L}}\} \\
[\![*e]\!]^{\Gamma}_{\mathcal{L}} &= [\![e]\!]^{\Gamma}_{\mathcal{A}}
\end{aligned}$$

Erweiterung der Semantik: Aexp(1)

$$[\![-]\!]_{\mathcal{A}} : \mathbf{Env} \to \mathbf{Aexp} \to \Sigma \rightharpoonup \mathbf{V}$$

$$[\![n]\!]^{\Gamma}_{\mathcal{A}} = \{(\sigma, n) \mid \sigma \in \Sigma\} \quad \text{für } n \in \mathbb{N}$$

$$[\![e]\!]^{\Gamma}_{\mathcal{A}} = \{(\sigma, v \mid (\sigma, l) \in [\![e]\!]^{\Gamma}_{\mathcal{L}}, (l, v) \in \sigma\} \qquad e \in \mathbf{Lexp} \text{ und } e \text{ kein Array-Typ}$$

$$[\![e]\!]^{\Gamma}_{\mathcal{A}} = \{(\sigma, l) \mid (\sigma, l) \in [\![e]\!]^{\Gamma}_{\mathcal{L}}\} \qquad e \in \mathbf{Lexp} \text{ und } e \text{ ist Array-Typ}$$

$$[\![\&e]\!]^{\Gamma}_{\mathcal{A}} = \{(\sigma, l) \mid (\sigma, l) \in [\![e]\!]^{\Gamma}_{\mathcal{L}}\}$$

- Es ist wichtig, die semantischen Funktionen $[\![e]\!]_{\mathcal{A}}$ und $[\![e]\!]_{\mathcal{L}}$ zu unterscheiden!

Erweiterung der Semantik: Aexp(1)

$$[\![-]\!]_{\mathcal{A}} : \mathbf{Env} \to \mathbf{Aexp} \to \Sigma \rightharpoonup \mathbf{V}$$

$$
\begin{aligned}
[\![n]\!]^{\Gamma}_{\mathcal{A}} &= \{(\sigma, n) \mid \sigma \in \Sigma\} \quad \text{für } n \in \mathbb{N} \\
[\![e]\!]^{\Gamma}_{\mathcal{A}} &= \{(\sigma, v \mid (\sigma, l) \in [\![e]\!]^{\Gamma}_{\mathcal{L}}, (l, v) \in \sigma\} && e \in \mathbf{Lexp} \text{ und } e \text{ kein Array-Typ} \\
[\![e]\!]^{\Gamma}_{\mathcal{A}} &= \{(\sigma, l) \mid (\sigma, l) \in [\![e]\!]^{\Gamma}_{\mathcal{L}}\} && e \in \mathbf{Lexp} \text{ und } e \text{ ist Array-Typ} \\
[\![\&e]\!]^{\Gamma}_{\mathcal{A}} &= [\![e]\!]^{\Gamma}_{\mathcal{L}}
\end{aligned}
$$

- Es ist wichtig, die semantischen Funktionen $[\![e]\!]_{\mathcal{A}}$ und $[\![e]\!]_{\mathcal{L}}$ zu unterscheiden!

Erweiterung der Semantik: Aexp(2)

$$[\![-]\!]_{\mathcal{A}} : \mathbf{Env} \to \mathbf{Aexp} \to \Sigma \rightharpoonup \mathbf{V}$$

$$\begin{aligned}
[\![a_0 + a_1]\!]^{\Gamma}_{\mathcal{A}} &= \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^{\Gamma}_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^{\Gamma}_{\mathcal{A}}\} \\
[\![a_0 - a_1]\!]^{\Gamma}_{\mathcal{A}} &= \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^{\Gamma}_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^{\Gamma}_{\mathcal{A}}\} \\
[\![a_0 * a_1]\!]^{\Gamma}_{\mathcal{A}} &= \{(\sigma, n_0 \cdot n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^{\Gamma}_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^{\Gamma}_{\mathcal{A}}\} \\
[\![a_0 / a_1]\!]^{\Gamma}_{\mathcal{A}} &= \{(\sigma, n_0 \div n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^{\Gamma}_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^{\Gamma}_{\mathcal{A}} \wedge n_1 \neq 0\}
\end{aligned}$$

Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto l_1, x \mapsto l_2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle l_1 \mapsto 10, l_2 \mapsto l_1 \rangle$$

$$[\![*x = 5 * a]\!]^{\Gamma}_{\mathcal{C}}(\sigma_1) = \sigma_1[[\![*x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1) \mapsto \sigma_1([\![5 * a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))]$$

Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto l_1, x \mapsto l_2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle l_1 \mapsto 10, l_2 \mapsto l_1 \rangle$$

$$\begin{aligned} [\![*x = 5 * a]\!]^{\Gamma}_{\mathcal{C}}(\sigma_1) &= \sigma_1[[\![*x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1) \mapsto \sigma_1([\![5 * a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \\ &= \sigma_1[[\![x]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \mapsto \sigma_1([\![5]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \cdot [\![a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \end{aligned}$$

Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto l_1, x \mapsto l_2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle l_1 \mapsto 10, l_2 \mapsto l_1 \rangle$$

$$\begin{aligned}
[\![*x = 5 * a]\!]^{\Gamma}_{\mathcal{C}}(\sigma_1) &= \sigma_1[[\![*x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1) \mapsto \sigma_1([\![5 * a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \\
&= \sigma_1[[\![x]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \mapsto \sigma_1([\![5]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \cdot [\![a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \\
&= \sigma_1[\sigma_1(\underbrace{[\![x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1)}_{\Gamma(x)}) \mapsto 5 \cdot \sigma_1(\underbrace{[\![a]\!]^{\Gamma}_{\mathcal{L}}}_{\Gamma(a)})]
\end{aligned}$$

Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto l_1, x \mapsto l_2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle l_1 \mapsto 10, l_2 \mapsto l_1 \rangle$$

$$\begin{aligned}
[\![*x = 5 * a]\!]^{\Gamma}_{\mathcal{C}}(\sigma_1) &= \sigma_1[[\![*x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1) \mapsto \sigma_1([\![5 * a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \\
&= \sigma_1[[\![x]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \mapsto \sigma_1([\![5]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \cdot [\![a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \\
&= \sigma_1[\sigma_1(\underbrace{[\![x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1)}_{\Gamma(x)}) \mapsto 5 \cdot \sigma_1(\underbrace{[\![a]\!]^{\Gamma}_{\mathcal{L}}}_{\Gamma(a)})] \\
&= \sigma_1[\sigma_1(l_2) \mapsto 5 \cdot \sigma_1(l_1)]
\end{aligned}$$

Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto l_1, x \mapsto l_2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle l_1 \mapsto 10, l_2 \mapsto l_1 \rangle$$

$$\begin{aligned}
[\![*x = 5 * a]\!]^{\Gamma}_{\mathcal{C}}(\sigma_1) &= \sigma_1[[\![*x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1) \mapsto \sigma_1([\![5 * a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \\
&= \sigma_1[[\![x]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \mapsto \sigma_1([\![5]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \cdot [\![a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1))] \\
&= \sigma_1[\sigma_1(\underbrace{[\![x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1)}_{\Gamma(x)}) \mapsto 5 \cdot \sigma_1(\underbrace{[\![a]\!]^{\Gamma}_{\mathcal{L}}}_{\Gamma(a)})] \\
&= \sigma_1[\sigma_1(l_2) \mapsto 5 \cdot \sigma_1(l_1)] \\
&= \sigma_1[l_1 \mapsto 5 \cdot 10] = \langle l_1 \mapsto 50, l_2 \mapsto l_1 \rangle
\end{aligned}$$

Arbeitsblatt 13.3: Pop-Quiz

Gegeben folgende Funktionen:

```
int f(int *x)
{
  int a;
  a= *x;
  *x= a+1;
  return a;
}
```

```
int a[3] = {0, 0, 0};
void g()
{
  int x= 1;
  a[x]= f(&x);
}
```

Was ist der Wert des Feldes a am Ende von g?

1. a == {0, 0, 1}
2. a == {0, 0, 2}
3. a == {0, 1, 0}
4. a == {0, 2, 0}

Korrektur der Semantik

- Das Pop-Quiz demonstriert den **Nichtdeterminismus** der C-Semantik.
  - Es ist **unbestimmt**, ob die linke oder rechte Seite der Zuweisung zuerst ausgewertet wird.
- Unsere Teilsprache soll den **deterministischen** Teil von C umfassen.
- Deshalb beim Funktionsaufruf mit Zuweisung links nur **zustandsfreie** Ausdrücke l:

$$\forall \sigma, \sigma'. [\![l]\!]^{\Gamma}_{\mathcal{L}}(\sigma) = [\![l]\!]^{\Gamma}_{\mathcal{L}}(\sigma')$$

- Gilt insbesondere für Bezeichner $l \in \mathbf{Idt}$, deshalb:

  Stmt $c ::= \ldots \mid \mathbf{Idt} = \mathbf{Idt}(a^*) \mid \ldots$

- Bei anderen Zuweisungen $l = e$ sind beide Seiten **zustandsabhängig**, aber frei von **Seiteneffekten**.

Arbeitsblatt 13.4: Kurze Semantik

Gegeben folgende Deklarationen:

```
struct p { int x;          int a;          struct p *q;
           int y; } p;
```

mit folgender Umgebung und Zustand

$$\Gamma \stackrel{def}{=} \langle p \mapsto l_1, a \mapsto l_2, q \mapsto l_3 \rangle, l_1 \neq l_2, \sigma_1 \stackrel{def}{=} \langle l_1.x \mapsto 3, l_1.y \mapsto 5, l_2 \mapsto 7, l_3 \mapsto l_1 \rangle$$

Berechnet die denotationale Semantik (erst für beliebige Zustände, dann für σ_1) von

$$[\![a = a + (*q).x]\!]^{\Gamma}_{C} = ?$$
$$[\![a = a + (*q).x]\!]^{\Gamma}_{C}(\sigma_1) = ?$$

Und jetzt?

- Zustand erweitert, so dass wir Zeiger modellieren können.
- Semantik entsprechend erweitert.
- Was machen wir mit dem Hoare-Kalkül, speziell der **Zuweisung**?

Und jetzt?

- Zustand erweitert, so dass wir Zeiger modellieren können.
- Semantik entsprechend erweitert.
- Was machen wir mit dem Hoare-Kalkül, speziell der **Zuweisung**?
- Vorherige Modellierung — Zuweisung durch Substitution modelliert — nicht mehr ausreichend.
- Daher: **explizite Zustandsprädikate**

Explizite Zustandsprädikate

- Zusicherungenen (**Assn**) sind zustandsabhängige Prädikate
  - Mit anderen Worten, Prädikate über Programmvariablen.
- Axiomatische Beschreibung des Zustandes erforderte neue Modellierung auf der Ebene der Prädikate
- Explizite Zustandsprädikate modellieren das Lesen und die Änderung des Zustandes **explizit** mit Operationen read und upd.

Explizite Zustandsprädikate

- Enthalten keine $*$ oder &, und unterscheiden **strikt** zwischen **Lexp** und **Aexp**.
- Erweiterung von **Aexpv** um read, neue Sorte **State** mit Operation upd:

$$\textbf{Assn}_s \quad b ::= \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 < a_2 \mid !\, b \mid b_1 \,\&\&\, b2 \mid \ldots$$

$$\textbf{Lexp}_s \quad l ::= \ldots \mid \sout{*a}$$

$$\textbf{Aexp}_s \quad a ::= \mathrm{read}(S, l) \mid \mathbf{Z} \mid \mathbf{C} \mid \sout{l} \mid \sout{\&l} \mid \ldots \mid e\,\textbf{@pre} \mid \ldots$$

$$\textbf{State} \quad S ::= s \mid r \mid \mathrm{upd}(S, l, e)$$

- Zustandsvariablen *StateVar*: aktueller Zustand s, Vorzustand r
- Im Gegensatz zur Semantik rechnen wir mit **symbolischen Namen**

Gleichungen für explizite Zustandsprädikate

- Für die Operationen read, updgelten folgende Gleichungen:

$$\forall l, v, \sigma.\ \mathsf{read}(\mathsf{upd}(\sigma, l, v), l) = v$$
$$\forall l, m, v, \sigma.\ l \neq m \Longrightarrow \mathsf{read}(\mathsf{upd}(\sigma, l, v), m) = \mathsf{read}(\sigma, m)$$
$$\forall l, v, w, \sigma.\ \mathsf{upd}(\mathsf{upd}(\sigma, l, v), l, w) = \mathsf{upd}(\sigma, l, w)$$
$$\forall l, m, v, w, \sigma.\ l \neq m \Longrightarrow \mathsf{upd}(\mathsf{upd}(\sigma, l, v), m, w) = \mathsf{upd}(\mathsf{upd}(\sigma, m, w), l, v)$$

- Diese Gleichungen sind **vollständig.**

Semantik expliziter Zustandsprädikate

- Semantik der expliziten Zustandsprädikate:

$$[\![.]\!]_{\mathcal{B}sp} : \mathbf{Env} \to \mathbf{Assn}_s \rightharpoonup (\Sigma \times (\Sigma \times \mathbf{V}_U)) \to \mathbb{B}$$
$$[\![.]\!]_{\mathcal{A}sp} : \mathbf{Env} \to \mathbf{Aexp}_s \rightharpoonup (\Sigma \rightharpoonup \mathbf{V})$$
$$[\![.]\!]_{\mathcal{L}sp} : \mathbf{Env} \to \mathbf{Lexp}_s \rightharpoonup (\Sigma \rightharpoonup \mathbf{Loc})$$

Hoare-Triple

- Floyd-Hoare-Tripel: $\Gamma \models \{P\}\, c\, \{Q \mid R\}$ mit $P, Q, R \in \mathbf{Assn}$ **Prädikate**
- Floyd-Hoare-Kalkül: $\Gamma \vdash \{P\}\, c\, \{Q \mid R\}$ mit $P, Q, R \in \mathbf{Assn}_s$ **explizite Zustandsprädikate**
- Wir brauchen also eine Übersetzung **Assn** nach $\mathbf{Assn}_s$ welche die Semantik erhält:

$$(-)^\dagger : \mathbf{Lexp} \to \mathbf{Lexp}_s \qquad \forall l \in \mathbf{Lexp}.\ [\![l]\!]^\Gamma_{\mathcal{L}} = [\![l^\dagger]\!]^\Gamma_{\mathcal{L}sp}$$

$$(-)^\# : \mathbf{Aexpv} \to \mathbf{Aexp}_s \qquad \forall a \in \mathbf{Aexpv}.\ [\![a]\!]^\Gamma_{\mathcal{A}} = [\![a^\#]\!]^\Gamma_{\mathcal{A}sp}$$

$$(-)^* : \mathbf{Assn} \to \mathbf{Assn}_s \qquad \forall a \in \mathbf{Assn}.\ [\![a]\!]^\Gamma_{\mathcal{B}} = [\![a^*]\!]^\Gamma_{\mathcal{B}sp}$$

Konversion in explizite Zustandsprädikaten

Alte Zuweisungsregel

$$\overline{\Gamma \vdash \{Q[e/x]\}\, x = e\, \{Q \mid R\}}$$

Umwandlung von Q, x, e in Zustandsprädikate:

- Eine **Lexp** l auf der rechten Seite e wird durch $\mathrm{read}(\sigma, l)$ ersetzt.
- & dient lediglich dazu, diese Konversion zu **verhindern**.
- $*$ **erzwingt** diese Konversion, auch auf der linken Seite x.
- Beispiel: $*a = *\&b;$

Formal: Konversion in Zustandsprädikate

$$
\begin{aligned}
(-)^{\dagger} &: \mathbf{Lexp} \to \mathbf{Lexp}_s \\
i^{\dagger} &= i \quad (i \in \mathbf{Idt}) \\
I.id^{\dagger} &= I^{\dagger}.id \\
I[e]^{\dagger} &= I^{\dagger}[e^{\#}] \\
*I^{\dagger} &= I^{\#}
\end{aligned}
\qquad
\begin{aligned}
(-)^{\#} &: \mathbf{Aexp} \to \mathbf{Aexp}_s \\
e^{\#} &= \mathrm{read}(s, e^{\dagger}) \quad (e \in \mathbf{Lexp}) \\
n^{\#} &= n \\
v^{\#} &= v \quad (v \text{ logische Variable}) \\
(\&e)^{\#} &= e^{\dagger} \\
(e_1 + e_2)^{\#} &= {e_1}^{\#} + {e_2}^{\#} \\
\backslash\mathbf{result}^{\#} &= \backslash\mathbf{result} \\
e\,\mathbf{@pre}^{\#} &= e^{\#}[s/r]
\end{aligned}
\qquad
\begin{aligned}
(-)^{*} &: \mathbf{Assn} \to \mathbf{Assn}_s \\
(\forall x.\, b)^{*} &= \forall x.\, b^{*} \\
(b_1 \wedge b_2)^{*} &= {b_1}^{*} \wedge {b_2}^{*} \\
(a_1 == a_2)^{*} &= {a_1}^{\#} = {a_2}^{\#} \\
&\vdots
\end{aligned}
$$

Angepasste Regeln des Hoare-Kalküls

$$\frac{}{\Gamma \vdash \{Q[\mathsf{upd}(s, x^\dagger, e^\#)/s]\}\, x = e\, \{Q \mid R\}}$$

- Beispiel 1: $(*x == 3)^*$
$$\begin{aligned} &= \mathsf{read}(s, (*x)^\dagger) = 3 \\ &= \mathsf{read}(s, x^\#) = 3 \\ &= \mathsf{read}(s, \mathsf{read}(s, x)) = 3 \end{aligned}$$

- Beispiel 2: $\Gamma \vdash \{P\}\, x = \&\, a\, \{*x = 3 \mid R\}$
$$\begin{aligned} P &= (\mathsf{read}(s, \mathsf{read}(s, x)) = 3)[\mathsf{upd}(s, x^\dagger, (\&\, a)^\#)/s] \\ &= (\mathsf{read}(s, \mathsf{read}(s, x)) = 3)[\mathsf{upd}(s, x, a^\dagger)/s] \\ &= \mathsf{read}(\mathsf{upd}(s, x, a), \mathsf{read}(\mathsf{upd}(s, \underline{x}, a), \underline{x})) = 3) \\ &= \mathsf{read}(\mathsf{upd}(s, \underline{x}, a), \underline{a}) = 3) \\ &= \mathsf{read}(s, a) = 3 \\ &= (a == 3)^* \end{aligned}$$

Weitere geänderte Regeln

$$\frac{}{\Gamma \vdash \{Q[e^{\#}/\backslash\textbf{result}]\}\ \textbf{return}\ \ e\,\{P \mid Q\}}$$

$$\frac{\Gamma(f) = \forall x_1, \ldots, x_n.\,(P, Q)}{\Gamma \vdash \{P[t_i{}^{\#}/x_i] \wedge y_i\ \textbf{@pre} = y_i\}\ l = f(t_1, \ldots, t_n)\,\{Q[t_i{}^{\#}/x_i][l^{\#}/\backslash\textbf{result}] \mid Q_R\}}$$

$$\frac{\Gamma \vdash \{P^*\}\ c\,\{\mathit{false} \mid Q^*\}}{\Gamma \vdash f(x_1, \ldots, x_n)/{**}\ \text{pre}\ P\ \text{post}\ Q\ {*}/\ \{ds\ c\}}$$

Rückwärtsrechnung mit Zeigern I

$$\text{awp}(\Gamma, f(x_1, \ldots, x_n)/{**}\ \text{pre}\ P\ \text{post}\ Q\ {*}/\ \{ds\ c\}) \stackrel{def}{=} \text{awp}(\Gamma, c, \mathit{false}, Q^*[x_i\ \textbf{@pre}\ /x_i])$$

$$\begin{aligned}\text{wvc}(\Gamma, f(x_1, \ldots, x_n)/{**}\ \text{pre}\ P\ \text{post}\ Q\ {*}/\ \{ds\ c\}) &\stackrel{def}{=} \{P^* \wedge x_i = x_i\ \textbf{@pre} \Longrightarrow P'\} \\ &\quad \cup\ \text{wvc}(\Gamma, c, \mathit{false}, Q^*[x_i\ \textbf{@pre}\ /x_i]) \\ P' &\stackrel{def}{=} \text{awp}(\Gamma, c, \mathit{false}, Q^*[x_i\ \textbf{@pre}\ /x_i])\end{aligned}$$

Sei $\Gamma(f) = \forall x_1, \ldots, x_n.\,(P, Q)$

$$\text{awp}(\Gamma, /{**}\ \textbf{const}\ \ R\ {*}/I = f(t_1, \ldots, t_n), U, U_R) \stackrel{def}{=} R^* \wedge P^*[{t_i}^*/x_i], I \notin FV(R)$$

$$\begin{aligned}\text{wvc}(\Gamma, /{**}\ \textbf{const}\ \ R\ {*}/I = f(t_1, \ldots, t_n), U, U_R) &\stackrel{def}{=} \{R^* \wedge Q[{t_i}^*/x_i][I/\backslash\textbf{result}] \longrightarrow U\}, \\ &\quad I \notin FV(R)\end{aligned}$$

Approximative schwächste Vorbedingung mit Zeigern II

$$\begin{aligned}
\mathrm{awp}(\Gamma, \{\,\}, Q, Q_R) &\stackrel{def}{=} Q \\
\mathrm{awp}(\Gamma, l = e, Q, Q_R) &\stackrel{def}{=} Q[\mathrm{upd}(s, l^\dagger, e^\#)/s] \\
\mathrm{awp}(\Gamma, c_1; c_2, Q, Q_R) &\stackrel{def}{=} \mathrm{awp}(\Gamma, c_1, \mathrm{awp}(c_2, Q, Q_R), Q_R) \\
\mathrm{awp}(\Gamma, \textbf{if } (b)\ c_0\ \textbf{ else }\ c_1, Q, Q_R) &\stackrel{def}{=} (b^* \wedge \mathrm{awp}(\Gamma, c_0, Q, Q_R)) \vee (\neg b^* \wedge \mathrm{awp}(c_1, Q, Q_R)) \\
\mathrm{awp}(\Gamma, /{*}{*}\ \{q\}\ {*}/, Q, Q_R) &\stackrel{def}{=} q^* \\
\mathrm{awp}(\Gamma, \textbf{while } (b)\ /{*}{*}\ \textbf{inv }\ i\ {*}/\ c, Q_R) &\stackrel{def}{=} i^* \\
\mathrm{awp}(\Gamma, \textbf{return }\ e, Q, Q_R) &\stackrel{def}{=} Q_R[e^\#/\backslash\textbf{result}] \\
\mathrm{awp}(\textbf{return}, Q, Q_R) &\stackrel{def}{=} Q_R
\end{aligned}$$

Approximative schwächste Vorbedingung mit Zeigern III

$$\begin{aligned}
\mathrm{wvc}(\Gamma, \{\,\}, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, l = e, Q, Q_R) &\stackrel{def}{=} \emptyset \\
\mathrm{wvc}(\Gamma, c_1; c_2, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c_1, \mathrm{awp}(\Gamma, c_2, Q, Q_R), Q_R) \cup \mathrm{wvc}(\Gamma, c_2, Q, Q_R) \\
\mathrm{wvc}(\Gamma, \textbf{if } (b)\ c_1\ \textbf{ else }\ c_2, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c_1, Q, Q_R) \cup \mathrm{wvc}(\Gamma, c_2, Q, Q_R) \\
\mathrm{wvc}(\Gamma, /{**}\ \{q\}\ {*}/, Q, Q_R) &\stackrel{def}{=} \{q^* \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \textbf{while } (b)\ /{**}\ \textbf{inv }\ i\ {*}/\ c, Q, Q_R) &\stackrel{def}{=} \mathrm{wvc}(\Gamma, c, i^*, Q_R) \cup \{i^* \wedge b^* \Longrightarrow \mathrm{awp}(\Gamma, c, i^*, Q_R)\} \\
&\qquad \cup \{i^* \wedge \neg b^* \Longrightarrow Q\} \\
\mathrm{wvc}(\Gamma, \textbf{return }\ e, Q, Q_R) &\stackrel{def}{=} \emptyset
\end{aligned}$$

Arbeitsblatt 13.5: Ein kurzes Beispiel

Betrachtet folgendes Beispiel:

```
void foo(){
 int x, y, z;
 x= 1;
 z= x;
 y= x;
 z= 5;
 // {0 < y}
}
```

1. Konvertiert das Prädikat $0 < y$ in ein explizites Zustandsprädikat.
2. Berechnet (rückwärts) die jeweils gültigen Zwischenzustände.
3. Vereinfacht nach jedem Schritt die Zwischenzustände.

Aliasing

- Das Beispiel mit Aliasing vom Anfang:

```
void foo(){
 int a, *x;

 /** { 5< 10 }
 /** { 5< read(upd(upd(upd(s, x, a), a, 0),                a     , 10), a) } */
 /** { 5< read(upd(upd(upd(s, x, a), a, 0), read(upd(s, x, a), x), 10), a) } */
 x= & a;
 /** { 5< read(upd(upd(s, a, 0), read(    s,         x), 10), a) } */
 /** { 5< read(upd(upd(s, a, 0), read(upd(s, a, 0), x), 10), a) } */
 a= 0;
 /** { 5< read(upd(s, read(s, x), 10), a) } */
 *x= 10;
 /** { 5< read(s, a) } */
}
```

- Aliasing wird **korrekt** behandelt.

Arbeitsblatt 13.6: Ein problematisches Beispiel

```
void foo(int *p)
/** post \result == 7; */
{
 int x;

 x= 7;
 *p= 99;
 return x;
}
```

Spezfikation des Speicherlayout

- Generelles Problem — was ist mit

```
void foo(int *p, int *q)
{ ... }
```

- Deutet auf ein Problem hin
- Entspricht einem “dangling pointer” (d.h. Pointer mit unspezifiziertem Wert)
- Können weder beweisen, dass $*p = *q$ noch $*p \neq *q$
- Muss (in den Vorbedingungen) spezifziert werden.
- Integration in die Logik: **separation logic**

Weitere Beispiele: Felder

```
int findmax(int a[], int a_len)
/** pre 0 \array(a, a_len) ∧ 0 < a_len;
    post ∀i. 0≤ i< a_len ⟶ a[i] ≤ \result ;
*/
{
 int x; int j;

 x= a[0]; j= 0;
 while (j< a_len)
   /** inv (∀i. 0≤ i< j ⟶ a[i]≤ x ∧ 0≤ j< a_len */ {
   if (a[j]< a_len) {
    x= a[j];
    }
   j= j+1;
   }
 return x;
}
```

Felder und Zeiger revisited

- In C sind Zeiger und Felder schwach spezifiziert
- Insbesondere:
  - a[j]= ∗(a+j) für a Array-Typ
  - Derefenzierung von ∗x nur definiert, wenn x "gültig" ist (d.h. auf ein Objekt zeigt) *C99 Standard*, §6.5.3.2(4)
- Bisher in den Hoare-Regeln ignoriert — **partielle** Korrektheit.
- Ist das sinnvoll? Nein, bekannte Fehlerquelle

Spezifikation von Zeigern und Feldern

Das Prädikat \valid(x)

$\backslash\textbf{valid}(x) \iff \text{read}(\sigma, x^\dagger)$ ist definiert

- Insbesondere: $\backslash\textbf{valid}(*x) \iff \text{read}(\sigma, \text{read}(\sigma, x))$ ist definiert.
- Felder als Parameter werden zu Zeigern konvertiert, deshalb müssen wir spezifizieren können, dass ein Zeiger ein Feld ist.
- $\backslash\textbf{array}(a, n)$ bedeutet: a ist ein Feld der Länge n, d.h.

$$\backslash\textbf{array}(a, n) \iff (\forall i.\, 0 \leq i < n \implies \backslash\textbf{valid}(a[i]))$$

- Gültigkeit kann abgeleitet werden:

$$\frac{\text{x= \&e}}{\backslash\text{valid(*x)}} \qquad \frac{\backslash\text{array(a, n)} \quad 0 \leq i \quad i < n}{\backslash\text{valid(a[i])}}$$

Was noch fehlt...

- **Vorwärtsrechung** mit expliziten Zustandsprädikaten.
  - Statt Existenzquantoren über Variablenwerte **unbestimmte Zwischenzustände** $\rho_1, \rho_2, \ldots$:

$$\frac{\rho_i \notin FV(P)}{\Gamma \vdash \{P\}\, x = e\, \{P[\rho_i/\sigma] \wedge \sigma = \mathrm{upd}(\rho_i, x^\dagger[\rho_i/\sigma], e^\#[\rho_i/\sigma]) \mid R\}}$$

  - Zwischenzustände sind **existenzquantifiziert**, d.h. das Prädikat gilt für **irgendeinen** Zustand ρ_i (aber für alle σ).
- Schwächste **Vorbedingung** und stärkste **Nachbedingung**:
  - Ergibt sich aus den Hoare-Regeln.
  - Erfordert durchgängige und agressive **Vereinfachung**.

Zusammenfassung

- Um Referenzen (Pointer) in C behandeln zu können, benötigen wir ein erweitertes **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 **expliziten Zustandsprädikaten**.
  - Zuweisung wird zu **Zustandsupdate**.
  - Problem:
    - Zustände werden **sehr groß**
    - Rückwärtsrechnung erzeugt schnell sehr große „unbestimmte“ Zustände, die nicht vereinfacht werden können
    - Hier ist Vorwärtsrechnung vorteilhaft

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül I
- Der Floyd-Hoare-Kalkül II: Invarianten
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Funktionen und Prozeduren I
- Funktionen und Prozeduren II
- Referenzen und Speichermodelle
- Ausblick und Rückblick