# Korrekte Software: Grundlagen und Methoden

## Vorlesung 12 vom 27.06.24

## Referenzen

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2024

# Organisatorisches

- Prüfungstermine:
- 04.07.2024, 16:00
- August/Anfang September

# Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten im Floyd-Hoare-Kalkül
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Funktionen und Prozeduren I
- Funktionen und Prozeduren II
- Referenzen
- 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 Typen: Zeiger (Pointer)

$$\textbf{Type}\ \ t ::= \textbf{void} \mid \textbf{char} \mid \textbf{int} \mid *t \mid \textbf{struct}\ \ \textbf{Idt}^{?}\ \{\textbf{Decl}^{+}\} \mid t\ \ \textbf{Idt}[a]$$

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

$$\textbf{Lexp}\ \ l ::= \textbf{Idt} \mid l[a] \mid l.\textbf{Idt} \mid *a$$

$$\textbf{Aexp}\ \ a ::= \textbf{Z} \mid \textbf{C} \mid l \mid \&l \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1/a_2$$

$$\textbf{Bexp}\ \ b ::= \ldots$$

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

$$\textbf{Stmt}\ \ c ::= \ldots$$

# Das Problem mit Zeigern

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

```
int a;
int *x;

x= &a;
a= 99;
// {a = 99}
*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**: $\mathbf{Loc} = \mathbf{Addr} \times \mathbb{N}$ (siehe Vorlesung 8)
  - Werte: $\mathbf{V} = \mathbb{Z}$
- Ansatz reicht nicht mehr.
- Locations müssen auch Werte sein:

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

und somit sind Zustände $\Sigma \stackrel{def}{=} \mathbf{Loc} \rightharpoonup (\mathbb{Z} + \mathbf{Loc})$

# Erweiterung der Semantik

- Denotation für **L-Ausdrücke**:

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

$$[\![x]\!]^{\Gamma}_{\mathcal{L}} = \{(\sigma, \langle \Gamma(x), 0 \rangle) \mid x \in \Gamma\}$$

$$[\![m[e]]\!]^{\Gamma}_{\mathcal{L}} = \{(\sigma, l + n \cdot \mathit{sizeof}(t)) \mid (\sigma, l) \in [\![m]\!]^{\Gamma}_{\mathcal{L}}, (\sigma, n) \in [\![e]\!]^{\Gamma}_{\mathcal{A}}, \Gamma \vdash m : t[x]\}$$

$$[\![m.a]\!]^{\Gamma}_{\mathcal{L}} = \{(\sigma, l + \mathit{off}_t(a)) \mid \Gamma \vdash m : t, (\sigma, l) \in [\![m]\!]^{\Gamma}_{\mathcal{L}}\}$$

$$[\![*e]\!]^{\Gamma}_{\mathcal{L}} = [\![e]\!]^{\Gamma}_{\mathcal{A}} \quad \text{mit } \Gamma \vdash e : *t$$

- Denotation für **Ausdrücke** ($m \in \mathbf{Lexp}$):

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

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

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

# Erweiterung der Semantik

- Denotation für **L-Ausdrücke**:

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

$$[\![x]\!]^{\Gamma}_{\mathcal{L}} = \{(\sigma, \langle \Gamma(x), 0 \rangle) \mid x \in \Gamma\}$$

$$[\![m[e]]\!]^{\Gamma}_{\mathcal{L}} = \{(\sigma, l + n \cdot \mathit{sizeof}(t)) \mid (\sigma, l) \in [\![m]\!]^{\Gamma}_{\mathcal{L}}, (\sigma, n) \in [\![e]\!]^{\Gamma}_{\mathcal{A}}, \Gamma \vdash m : t[x]\}$$

$$[\![m.a]\!]^{\Gamma}_{\mathcal{L}} = \{(\sigma, l + \mathit{off}_t(a)) \mid \Gamma \vdash m : t, (\sigma, l) \in [\![m]\!]^{\Gamma}_{\mathcal{L}}\}$$

$$[\![*e]\!]^{\Gamma}_{\mathcal{L}} = [\![e]\!]^{\Gamma}_{\mathcal{A}} \quad \text{mit } \Gamma \vdash e : *t$$

- Denotation für **Ausdrücke** ($m \in \mathbf{Lexp}$):

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

$$[\![\&m]\!]^{\Gamma}_{\mathcal{A}} = [\![m]\!]^{\Gamma}_{\mathcal{L}}$$

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

# Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto 1, x \mapsto 2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle (1,0) \mapsto 10, (2,0) \mapsto (1,0) \rangle$$

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

# Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto 1, x \mapsto 2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle (1,0) \mapsto 10, (2,0) \mapsto (1,0) \rangle$$

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

# Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto 1, x \mapsto 2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle (1,0) \mapsto 10, (2,0) \mapsto (1,0) \rangle$$

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

# Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto 1, x \mapsto 2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle (1,0) \mapsto 10, (2,0) \mapsto (1,0) \rangle$$

$$\begin{aligned}
[\![*x = 5 * a]\!]^{\Gamma}_{\mathcal{C}}(\sigma_1) &= \sigma_1[[\![*x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1) \mapsto [\![5 * a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1)] \\
&= \sigma_1[[\![x]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \mapsto [\![5]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \cdot [\![a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1)] \\
&= \sigma_1[\sigma_1(\underbrace{[\![x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1)}_{\Gamma(x)}) \mapsto 5 \cdot \sigma_1(\underbrace{[\![a]\!]^{\Gamma}_{\mathcal{L}}}_{\Gamma(a)})] \\
&= \sigma_1[\sigma_1(2,0) \mapsto 5 \cdot \sigma_1(1,0)]
\end{aligned}$$

# Beispiel

```
int a, *x;

*x= 5*a;
```

Gegeben folgende Werte, was ist die Semantik?

$$\Gamma \stackrel{def}{=} \langle a \mapsto 1, x \mapsto 2 \rangle$$

$$\sigma_1 \stackrel{def}{=} \langle (1,0) \mapsto 10, (2,0) \mapsto (1,0) \rangle$$

$$\begin{aligned}
[\![*x = 5 * a]\!]^{\Gamma}_{\mathcal{C}}(\sigma_1) &= \sigma_1[[\![*x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1) \mapsto [\![5 * a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1)] \\
&= \sigma_1[[\![x]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \mapsto [\![5]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1) \cdot [\![a]\!]^{\Gamma}_{\mathcal{A}}(\sigma_1)] \\
&= \sigma_1[\sigma_1(\underbrace{[\![x]\!]^{\Gamma}_{\mathcal{L}}(\sigma_1)}_{\Gamma(x)}) \mapsto 5 \cdot \sigma_1(\underbrace{[\![a]\!]^{\Gamma}_{\mathcal{L}}}_{\Gamma(a)})] \\
&= \sigma_1[\sigma_1(2,0) \mapsto 5 \cdot \sigma_1(1,0)] \\
&= \sigma_1[(1,0) \mapsto 5 \cdot 10] = \langle (1,0) \mapsto 50, (2,0) \mapsto (1,0) \rangle
\end{aligned}$$

# Arbeitsblatt 12.1: Aliasing-Semantik

```
{
  int a;
  int *x;

  // (0)
  x= &a;
  a= 99;
  // (1)
  *x= 7;
  // (2)
}
```

Berechnet mit Hilfe der formalen Semantik:

1. Die Umgebung $\Gamma$ an der Stelle (0)
2. Den Zustand $\sigma_1$ an der Stelle (1)
3. Den Zustand $\sigma_2$ an der Stelle (2)

# Arbeitsblatt 12.2: Pop-Quiz

Gegeben folgende Funktionen:

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

Was ist der Wert von f(&x)?

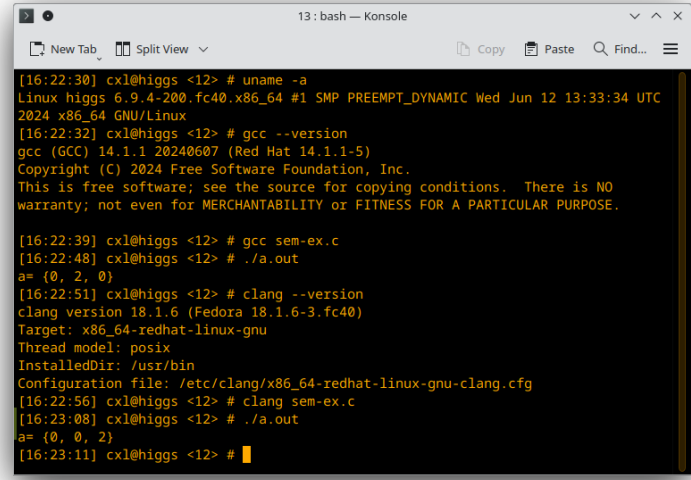
1. f(&x) == 1
2. f(&x) == 2

```
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}

# Ergebnis des Pop-Quiz



# 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.
  - In C ist die Zuweisung ein binärer Operator, keine Anweisung.
- 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 $l \in \mathbf{ldt}$, deshalb:

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

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

# Arbeitsblatt 12.3: Kurze Semantik

Gegeben folgende Deklarationen:

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

mit folgender Umgebung und Zustand

$$\Gamma \stackrel{def}{=} \langle p \mapsto 2, a \mapsto 3, q \mapsto 4 \rangle, \sigma_1 \stackrel{def}{=} \langle (2,0) \mapsto 3, (2,1) \mapsto 5, (3,0) \mapsto 7, (4,0) \mapsto (2,0) \rangle$$

Berechnet die denotationale Semantik

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

# 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.
- Explizite Zustandsprädikate modellieren das Lesen und die Änderung des Zustandes **explizit** mit Operationen read und upd.
- Vereinfachungsregeln analog zur Substitution.

# Explizite Zustandsprädikate

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

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

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

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

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

- Zustandsvariable $s$ (logische Variable): aktueller Zustand
- 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:

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

# Hoare-Triple

- Floyd-Hoare-Tripel:

$$\Gamma \models \{P\}\, c\, \{Q \mid R\} \text{ mit } P, Q, R \in \mathbf{Assn} \text{ Prädikate}$$

- Floyd-Hoare-Kalkül:

$$\Delta \vdash \{P\}\, c\, \{Q \mid R\} \text{ mit } P, Q, R \in \mathbf{Assn}_s \text{ explizite Zustandsprädikate}$$

- Wir brauchen also eine Übersetzung **Assn** nach $\mathbf{Assn}_s$ welche die Semantik erhält:

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

# Konversion in explizite Zustandsprädikaten

**Alte Zuweisungsregel**

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

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: $*\mathrm{a} = *\&\mathrm{b};$

# Formal: Konversion in Zustandsprädikate

$$(-)^{\dagger} : \mathbf{Lexp} \to \mathbf{Lexp}_s$$

$$i^{\dagger} = i \quad (i \in \mathbf{Idt})$$

$$l.id^{\dagger} = l^{\dagger}.id$$

$$l[e]^{\dagger} = l^{\dagger}[e^{\#}]$$

$$*l^{\dagger} = l^{\#}$$

$$(-)^{\#} : \mathbf{Aexp} \to \mathbf{Aexp}_s$$

$$e^{\#} = \mathrm{read}(s, e^{\dagger}) \quad (e \in \mathbf{Lexp})$$

$$n^{\#} = n$$

$$v^{\#} = v \quad (v \text{ logische Variable})$$

$$(\&e)^{\#} = e^{\dagger}$$

$$(e_1 + e_2)^{\#} = {e_1}^{\#} + {e_2}^{\#}$$

$$\backslash\mathbf{result}^{\#} = \backslash\mathbf{result}$$

$$(-)^{*} : \mathbf{Assn} \to \mathbf{Assn}_s$$

$$(\forall x.\, b)^{*} = \forall x.\, b^{*}$$

$$(b_1 \,\&\&\, b_2)^{*} = {b_1}^{*} \,\&\&\, {b_2}^{*}$$

$$(a_1 == a_2)^{*} = {a_1}^{\#} == {a_2}^{\#}$$

$$\vdots$$

# Angepasste Regeln des Hoare-Kalküls

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

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

- Beispiel 2: $\Delta \vdash \{P\}\, x = \&\, a\, \{*x = 3 \mid R\}$

$$\begin{aligned} P &= (\mathrm{read}(s, \mathrm{read}(s, x)) = 3)[\mathrm{upd}(s, x^\dagger, (\&\, a)^\#)/s] \\ &= (\mathrm{read}(s, \mathrm{read}(s, x)) = 3)[\mathrm{upd}(s, x, a^\dagger)/s] \\ &= \mathrm{read}(\mathrm{upd}(s, x, a), \mathrm{read}(\mathrm{upd}(s, \underline{x}, a), \underline{x})) = 3) \\ &= \mathrm{read}(\mathrm{upd}(s, \underline{x}, a), \underline{a}) = 3) \\ &= \mathrm{read}(s, a) = 3 \\ &= (a == 3)^* \end{aligned}$$

# Weitere geänderte Regeln

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

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

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

# Rückwärtsrechnung mit Zeigern I

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

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

$$\text{Sei}\ \Delta(f) = \forall x_1,\ldots,x_n.\,(P,Q)$$

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

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

# Approximative schwächste Vorbedingung mit Zeigern II

$$\mathrm{awp}(\Delta, \{\,\}, Q, Q_R) \stackrel{def}{=} Q$$

$$\mathrm{awp}(\Delta, l = e, Q, Q_R) \stackrel{def}{=} Q[\mathrm{upd}(s, l^\dagger, e^\#)/s]$$

$$\mathrm{awp}(\Delta, c_1; c_2, Q, Q_R) \stackrel{def}{=} \mathrm{awp}(\Delta, c_1, \mathrm{awp}(\Delta, c_2, Q, Q_R), Q_R)$$

$$\mathrm{awp}(\Delta, \textbf{if } (b)\ c_0\ \textbf{ else }\ c_1, Q, Q_R) \stackrel{def}{=} (b^* \wedge \mathrm{awp}(\Delta, c_0, Q, Q_R)) \vee (\neg b^* \wedge \mathrm{awp}(\Delta, c_1, Q, Q_R))$$

$$\mathrm{awp}(\Delta, /{**}\ \{q\}\ {*}/, Q, Q_R) \stackrel{def}{=} q^*$$

$$\mathrm{awp}(\Delta, \textbf{while } (b)\ /{**}\ \textbf{inv }\ i\ {*}/\ c, Q_R) \stackrel{def}{=} i^*$$

$$\mathrm{awp}(\Delta, \textbf{return }\ e, Q, Q_R) \stackrel{def}{=} Q_R[e^\#/\backslash\textbf{result}]$$

$$\mathrm{awp}(\Delta, \textbf{return}, Q, Q_R) \stackrel{def}{=} Q_R$$

# Approximative schwächste Vorbedingung mit Zeigern III

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

# Arbeitsblatt 12.4: 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 12.5: 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(l)

$\backslash\mathbf{valid}(l) \iff \exists x.\, x = \mathrm{read}(\sigma, l^{\dagger})$ für $l \in \mathbf{Lexp}$

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

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

- Gültigkeit kann abgeleitet werden:

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

# 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
- Invarianten im Floyd-Hoare-Kalkül
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Funktionen und Prozeduren I
- Funktionen und Prozeduren II
- Referenzen
- Ausblick und Rückblick