Formale Methoden der Softwaretechnik II

Christoph Lüth
http://www.informatik.uni-bremen.de/~cxl

Sommersemester'06

Vorlesung vom 25.04.06: Einführung

Organisatorisches

- Veranstalter: Christoph Lüth
  cxl@informatik.uni-bremen.de
  http://www.informatik.uni-bremen.de/~cxl
  MZH 8050, Tel. 7585

- Termine: Vorlesung: Dienstag 10– 12 MZH 8090
  Übung: Donnerstag 9 s.t. – 10 MZH 6240

Struktur

- Vorlesung wie gehabt
- Übungen:
  - Beginn 04.05.06 (nächste Woche)
  - Eine längere Übung: Implementation eines Verifikationswerkzeuges
  - Teilaufgaben pro Woche
  - Lösungen vorstellen

Inhalt

- Statt breiter Übersicht (FMSE I) Schwerpunkte.
- Programmverifikation:
- Entwicklung korrekter Programme:
- Blick in die Industrie: Anwendungen

Inhalt — Programmverifikation

- Floyd-Hoare-Logik (zur Wiederholung),
- Prädikatentransformer und schwächste Vorbedingungen,
- Generation von Verifikationsbedingungen,
- Modellierung von imperativen Programmen,
- Verifikationswerkzeuge.

Inhalt — Entwicklung korrekter Programme

- Entwicklung von korrekten Programmen durch Transformation,
- Algorithmendesign,
- Transformationswerkzeuge.

Floyd-Hoare-Logik

- Vor- und Nachbedingungen:

$$\{A\}c\{B\}$$

  Bedeutung: jeder Zustand, der A erfüllt, ist nach Termination von c ein Zustand, der B erfüllt.

- Hoare-Kalkül: Beweisregeln

Regeln des Hoare-Kalküls

$$\frac{}{\{A\}\texttt{skip}\{A\}}\,Skip \qquad \frac{}{\{B[a/X]\}X := a\{B\}}\,Assgn$$

$$\frac{\{A\}c_0\{C\} \quad \{C\}c_1\{B\}}{\{A\}c_0; c_1\{B\}}\,Seq$$

$$\frac{\{A \wedge b\}c_0\{B\} \quad \{A \wedge \neg b\}c_1\{B\}}{\{A\}\texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \texttt{ fi}\{B\}}\,Cond$$

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\texttt{while } b \texttt{ do } c \texttt{ od}\{A \wedge \neg b\}}\,While$$

$$\frac{(A \Longrightarrow A') \quad \{A'\}c\{B'\} \quad (B' \Longrightarrow B)}{\{A\}c\{B\}}\,Impl$$

Beispiel

```
int r= x;
int q= 0;
while (r >= y) do
  r= r-y;
  q= q+1
  od
```

Fazit Hoare-Kalkül

- Umständlich zu ‘rechnen’
- Vermischung von Logik und Berechnung
  - z.B. Bedingung bei `if`, Invariante bei `while`
  - Dadurch: Seiteneffekte problematisch

```
if (x++ == 0) then p else q fi;
```

Nächste Vorlesung

- Korrektheit des Hoare-Kalküls
- Dafür: Semantik durch Prädikatentransformer
- Damit: systematische Berechnung von Vorbedingungen

Vorlesung vom 02.05.06: Korrektheit des Hoare-Kalküls

Wo sind wir?

- Letze Woche: Erinnerungen an den Hoare-Kalkül
- Diese Woche: Korrektheit des Hoare-Kalküls
- Nächste Woche: Vollständigkeit des Hoare-Kalküls

Fahrplan

Korrektheit des Hoare-Kalküls: $\{A\}c\{B\}$ ableitbar wenn gültig.

1. Genaue Definition von $\{A\}c\{B\}$: Vor- & Nachbedingung: Zusicherungen
2. Definition von Ableitbarkeit $\vdash \{A\}c\{B\}$
3. Definition von Gültigkeit $\models \{A\}c\{B\}$:
   - Gültigkeit von Zusicherungen $\sigma \models A$ in Zustand σ
   - $\models \{A\}c\{B\}$ gdw: wenn A in Zustand σ gilt, dann gilt B im Zustand nach Ausführung von c
   - Dazu: Zustandsübergang $\longrightarrow$ denotationelle Semantik

Die Sprache IMP (Wiederholung)

- Zahlen: **N** $= \{0, 1, 2, 3, \ldots\}$
- Variablen: **Loc**
- Arithmetische Ausdrücke: $a \in$ **Aexp**

  $a ::= n \mid x \mid a \texttt{ + } a \mid a \texttt{ - } a \mid a \texttt{ * } a$ $\qquad n \in$ **N**$, x \in$ **Loc**
- Boolsche Ausdrücke: $b \in$ **Bexp**

  $b ::= \texttt{True} \mid \texttt{False} \mid a \texttt{ = } a \mid a \texttt{ <= } a \mid \texttt{not } b \mid b \texttt{ and } b \mid b \texttt{ or } b$
- Anweisungen:

  Com$c ::= \texttt{skip} \mid x := a \mid c; c \mid$
  $\texttt{if } b \texttt{ then } c \texttt{ else } c \texttt{ fi} \mid \texttt{while } b \texttt{ do } c \texttt{ od}$

Bedeutung von IMP

- Operationale Semantik: ⟵ FMSE I
  - Syntaxgesteuerte Ableitungsregeln (Zustandsübergang)
  - Für: Sprachdefinition, Compilerverifikation
- Denotationale Semantik: ⟵ FMSE I
  - Abbildung $\mathcal{D} : \mathbf{Com} \to \mathcal{S}$ in semantischen Bereich
  - Für: Programmentwicklung und Programmtransformation
- Axiomatische Semantik:
  - Vor- und Nachbedingungen für jede Anweisung
  - Für: Programmverifikation

Denotationale Semantik für IMP

- Programm als Funktion zwischen Systemzuständen
- Systemzustand: $\Sigma \stackrel{def}{=} \mathbf{Loc} \rightharpoonup \mathbb{N}$
- Semantische Funktionen:
  - Arithmetische Ausdrücke: $\mathcal{E} : \mathbf{Aexp} \to (\Sigma \to \mathbb{N})$
  - Boolsche Ausdrücke: $\mathcal{B} : \mathbf{Bexp} \to (\Sigma \to \mathit{Bool})$
  - Anweisungen: $\mathcal{D} : \mathbf{Com} \to (\Sigma \rightharpoonup \Sigma)$

Denotationale Semantik für IMP: Anweisungen

$$\mathcal{C}[\![\texttt{skip}]\!] \stackrel{def}{=} \lambda\sigma^{\Sigma}.\sigma$$

$$\mathcal{C}[\![X := a]\!] \stackrel{def}{=} \lambda\sigma^{\Sigma}.\sigma[\mathcal{E}[\![a]\!]\sigma/X]$$

$$\mathcal{C}[\![c_0; c_1]\!] \stackrel{def}{=} \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!]$$

$$\mathcal{C}[\![\texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \texttt{ fi}]\!] \stackrel{def}{=} \lambda\sigma^{\Sigma}.\begin{cases} \mathcal{C}[\![c_0]\!]\sigma & \mathcal{B}[\![b]\!]\sigma = true \\ \mathcal{C}[\![c_1]\!]\sigma & \mathcal{B}[\![b]\!]\sigma = false \end{cases}$$

$$\mathcal{C}[\![\texttt{while } b \texttt{ do } c \texttt{ od}]\!] \stackrel{def}{=} fix\ \Gamma$$

$$\text{mit } \Gamma(\phi) \stackrel{def}{=} \lambda\sigma^{\Sigma}.\begin{cases} \phi \circ \mathcal{C}[\![c]\!]\sigma & \mathcal{B}[\![b]\!]\sigma = true \\ \sigma & \mathcal{B}[\![b]\!]\sigma = false \end{cases}$$

Zusicherungen

- Warum nicht einfach boolsche Ausdrücke?

Zusicherungen

- Warum nicht einfach boolsche Ausdrücke?
- Menge von Variablen: **Intvar**(logische Variablen, vgl. **Loc**)
- Erweiterte arithmetische Ausdrücke **Aexpv**
  $a ::= i \mid n \mid x \mid a \texttt{ + } a \mid a \texttt{ - } a \mid a \texttt{ * } a \quad i \in \textbf{Intvar}, n \in \textbf{N}, x \in \textbf{Loc}$
- Zusicherungen sind erweiterte boolsche Ausdrücke **Assn**:
  $b ::= \texttt{True} \mid \texttt{False} \mid a \texttt{ = } a \mid a \texttt{ <= } a \mid \texttt{not } b \mid b \texttt{ and } b \mid$
  $b \texttt{ or } b \mid b \Rightarrow b \mid \forall i.b \mid \exists i.b$
- Substitution:
  - $a\,[a'/i]$ ($a' \in \textbf{Aexpv}, a \in \textbf{Aexp}, i \in \textbf{Intvar}$)
  - $b\,[a/i]$ ($b \in \textbf{Assn}, a \in \textbf{Aexp}, i \in \textbf{Intvar}$)
  - Definiert durch strukturelle Induktion.

Semantik von Zusicherungen

- Problem: Wie interpretieren wir Variablen?

Semantik von Zusicherungen

- Problem: Wie interpretieren wir Variablen?
- Eine Interpretation ist I : **Intvar** $\to \mathbb{N}$
- Auswertungen mit Interpretationen:
  - Funktion $\mathcal{A}$: **Aexpv** $\to$ (**Intvar** $\to \mathbb{N}$) $\to$ ($\Sigma \to \mathbb{N}$)
  - Prädikat $\sigma \models^I A$ für $A \in$ **Assn**

Semantik von Zusicherungen

- $\mathcal{A}$ definiert durch strukturelle Induktion:

$$\begin{aligned}
\mathcal{A}[\![n]\!]^I_\sigma &= n \\
\mathcal{A}[\![X]\!]^I_\sigma &= \sigma(X) \\
\mathcal{A}[\![i]\!]^I_\sigma &= I(i) \\
\mathcal{A}[\![a_0 \texttt{ + } a_1]\!]^I_\sigma &= \mathcal{A}[\![a_0]\!]^I_\sigma + \mathcal{A}[\![a_1]\!]^I_\sigma \\
&\ldots
\end{aligned}$$

- Lemma: Für $a, a_0 \in$ **Aexpv**, $X \in$ **Loc** und alle $\sigma \in \Sigma$:

$$\mathcal{A}[\![a_0[a/X]]\!]^I_\sigma = \mathcal{A}[\![a_0]\!]^I_{\sigma[\mathcal{A}[\![a]\!]^I_\sigma/X]}$$

- Lemma: Für $b \in$ **Assn**, $X \in$ **Loc**, $a \in$ **Aexp** und alle $\sigma \in \Sigma$:

$$\sigma \models^I B[a/X] \iff \sigma[\mathcal{E}[\![a]\!]\sigma/X] \models^I B$$

Semantik von Zusicherungen

Def (Erfüllbarkeit von Zusicherungen): Mit $\sigma : \Sigma_\perp$, $I :$ **Intvar** $\to \mathbb{N}$:

$$
\begin{array}{lll}
\sigma \models^I \texttt{True} & & \\
\sigma \models^I a_0 \texttt{ = } a_1 & \text{if} & \mathcal{A}[\![a_0]\!]^I_\sigma = \mathcal{A}[\![a_1]\!]^I_\sigma \\
\sigma \models^I a_0 \texttt{ <= } a_1 & \text{if} & \mathcal{A}[\![a_0]\!]^I_\sigma \leq \mathcal{A}[\![a_1]\!]^I_\sigma \\
\sigma \models^I b_0 \texttt{ and } b_1 & \text{if} & \sigma \models^I b_0 \wedge \sigma \models^I b_0 \\
\sigma \models^I b_0 \texttt{ or } b_1 & \text{if} & \sigma \models^I b_0 \vee \sigma \models^I b_0 \\
\sigma \models^I \texttt{not } b & \text{if} & \neg\sigma \models^I b \\
\sigma \models^I b_0 \Rightarrow b_1 & \text{if} & \sigma \models^I b_0 \Longrightarrow \sigma \models^I b_0 \\
\sigma \models^I \forall i.b & \text{if} & \sigma \models^{I[n/i]} b \text{ for all } n \in N \\
\sigma \models^I \exists i.b & \text{if} & \sigma \models^{I[n/i]} b \text{ for some } n \in N \\
\perp \models^I a & &
\end{array}
$$

Gültigkeit

Def (Satisfaction Relation, Gültigkeit): Für Interpretation I, $\sigma \in \Sigma_\perp$:

$$\sigma \models^I \{A\}c\{B\} \iff (\sigma \models^I A \Longrightarrow \mathcal{C}[\![c]\!]\sigma \models^I B)$$

$$\models^I \{A\}c\{B\} \iff \forall \sigma : \Sigma_\perp . \sigma \models^I \{A\}c\{B\}$$

$$\models \{A\}c\{B\} \iff \forall I . \models^I \{A\}c\{B\}$$

Analog für Zusicherungen A ($\models A$ etc.)

NB. Gültigkeit im Gegensatz zum herkömmlichen Modellbegriff.

Beispiel: $\models A \Longrightarrow B$, $\models \{A\}c\{B\}$.

Korrektheit des Hoare-Kalküls

Def (Herleitbarkeit): $\vdash \{A\}c\{B\}$ durch die Regeln des Hoare-Kalküls

Satz (Korrektheit des Hoare-Kalküls):

$$\text{Wenn } \vdash \{A\}c\{B\}, \text{ dann } \models \{A\}c\{B\}.$$

Beweis: Fallunterscheidung über den Regeln.

- $c \equiv X := a$
- $c \equiv c_0; c_1$
- $c \equiv \texttt{if}\ b\ \texttt{then}\ c_0\ \texttt{else}\ c_1\ \texttt{fi}$
- $c \equiv \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}$
- Implikationsregel

Zusammenfassung

- Heute: Korrektheit des Hoare-Kalküls
- Dazu:
  - Zusicherungen **Assn**, syntaktische Herleitbarkeit
  - Interpretationen, semantische Gültigkeit
  - Korrektheit: Herleitbarkeit impliziert Gültigkeit
- Nächste Woche: Vollständigkeit — Gültigkeit impliziert Herleitbarkeit

Vorlesung vom 09.05.06: Vollständigkeit des Hoare-Kalküls

Wo sind wir?

- Voretze Woche: Erinnerungen an den Hoare-Kalkül
- Letzte Woche: Korrektheit des Hoare-Kalküls
- Diese Woche: Vollständigkeit des Hoare-Kalküls
- Nächste Woche: Verifikationsbedingungen und Prädikatentransformer

Vollständigkeit des Hoare-Kalküls

- Korrektheit: wir können nur wahre Folgerungen ableiten
- Vollständigkeit: wir können alle wahren Folgerungen ableiten
- Satz (Unvollständgkeit des Hoare-Kalküls): Es kann keinen korrekten, vollständigen Beweiskalkül für den Hoare-Kalkül geben.

  Beweis: Vollständiger Beweiskalkül würde Termination entscheiden: $\models \{\texttt{True}\}c\{\texttt{False}\}$ gilt gdw. c divergiert.
- Satz (Unvollständigkeit für **Assn**): Es kann keinen korrekten, vollständigen Beweiskalkül für **Assn**geben.

Vollständigkeit des Hoare-Kalküls

- Korrektheit: wir können nur wahre Folgerungen ableiten
- Vollständigkeit: wir können alle wahren Folgerungen ableiten
- Satz (Unvollständgkeit des Hoare-Kalküls): Es kann keinen korrekten, vollständigen Beweiskalkül für den Hoare-Kalkül geben.

  Beweis: Vollständiger Beweiskalkül würde Termination entscheiden: $\models \{\texttt{True}\}c\{\texttt{False}\}$ gilt gdw. c divergiert.
- Satz (Unvollständigkeit für **Assn**): Es kann keinen korrekten, vollständigen Beweiskalkül für **Assn**geben.
- Aber relative Vollständigkeit beweisbar:
  Wenn $\models \{A\}c\{B\}$, dann $\vdash \{A\}c\{B\}$.

Schwächste Vorbedingung

Def (Schwächste Vorbedingung): Für $c \in$ **Com**, $B \in$ **Assn**, Interpret. I:

$$wp^I[\![c, b]\!] = \{\sigma \in \Sigma_\perp \mid \mathcal{C}[\![c]\!]\sigma \models^I B\}$$

- Wenn es A_0 gibt so dass für alle I $A_0^I = wp^I[\![c, b]\!]$, dann $\models \{A\}c\{B\}$ gdw $\models A \Longrightarrow A_0$.
- Gibt es so ein A_0?

Def (Expressivität): **Assn** ist expressiv, wenn es so ein A_0 für gibt.

Gödel's β-Prädikat

- Modulus in **Assn**: $x = a \mod b$ ist
  $a \geq 0 \wedge b \geq 0 \wedge \exists k.(k \geq 0 \wedge k * b \geq a \wedge (k + 1) * b > a \wedge x = a - (k * b))$
- Def (Prädikat β):

$$\beta(a, b, i, x) \Longleftrightarrow x = a \mod (1 + (1 + i) * b)$$

Gödel's β-Prädikat

- Modulus in **Assn**: $x = a \mod b$ ist
  $a \geq 0 \wedge b \geq 0 \wedge \exists k.(k \geq 0 \wedge k * b \geq a \wedge (k + 1) * b > a \wedge x = a - (k * b)$
- Def (Prädikat β):

$$\beta(a, b, i, x) \Longleftrightarrow x = a \mod (1 + (1 + i) * b)$$

- Lemma: Für Folge $n_0, \ldots, n_k$ gibt es $n, m \in \mathbb{N}$ so dass für alle j, $0 \leq j \leq k$ und x:

$$\beta(n, m, j, x) \Longleftrightarrow x = n_j$$

- Kodierung von Folgen
- Technisches Detail: **IMP** hat ganze Zahlen (aber $\mathbb{N} \cong \mathbb{Z}$)

Expressivität von Assn

Satz (Expressivität): Für $c \in$ **Com**, $B \in$ **Assn** gibt es $w[\![c, b]\!] \in$ **Assn** so dass für alle I

$$wp^I[\![c, B]\!] = w[\![c, B]\!]$$

Beweis: Konstruktion von $w[\![c, B]\!]$ durch Induktion über c:

- $c \equiv \texttt{skip}$
- $c \equiv X := a$
- $c \equiv c_0; c_1$
- $c \equiv \texttt{if}\ b\ \texttt{then}\ c_0\ \texttt{else}\ c_1\ \texttt{fi}$
- $c \equiv \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}$

Expressivität von Assn

Lemma: Für $w \equiv \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}$:

$$\begin{aligned}\mathcal{C}[\![w]\!]\sigma = \sigma' \text{ gdw } & \mathcal{B}[\![b]\!]\sigma = \mathit{false} \wedge \sigma = \sigma' \\ & \vee \\ & \exists \sigma_0, \ldots, \sigma_n \in \Sigma. \\ & \sigma = \sigma_0 \wedge \sigma' = \sigma_n \wedge \mathcal{B}[\![b]\!]\sigma_n = \mathit{false} \wedge \\ & \forall i. 0 \leq i \leq n \Longrightarrow \mathcal{B}[\![b]\!]\sigma_i = \mathit{true} \wedge \mathcal{C}[\![c]\!]\sigma_i = \sigma_{i+1}\end{aligned}$$

Vollständigkeit

Lemma: Für $c \in$ **Com**, $B \in$ **Assn**, sei $w[\![c, B]\!]$ eine schwächste Vorbedingung, dann $\vdash \{w[\![c, b]\!]\}c\{B\}$.

Beweis: Induktion über c. $\square$

Vollständigkeit

Lemma: Für $c \in$ **Com**, $B \in$ **Assn**, sei $w[\![c, B]\!]$ eine schwächste Vorbedingung, dann $\vdash \{w[\![c, b]\!]\}c\{B\}$.

Beweis: Induktion über c. $\square$

Satz (Relative Vollständigkeit des Hoare-Kalküls):

$$\text{Wenn } \models \{A\}c\{B\}, \text{ dann } \vdash \{A\}c\{B\}.$$

Beweis: Wenn $\models \{A\}c\{B\}$, dann $\vdash \{w[\![c, b]\!]\}c\{B\}$, also mit $\models A \Longrightarrow w[\![c, b]\!]$ folgt $\models \{A\}c\{B\}$ (Folgerungsregel). $\square$

Zusammenfassung

- Allgemeine Vollständigkeit unmöglich (wenn Kalkül konsistent), da **Assn** zu mächtig.
- Daher: Beweis der relativen Vollständigkeit
- Beweise beruht auf
  - schwächster Vorbedingung $wp^I[\![c, B]\!]$: Menge der Zustände, die B nach Ausführung von c wahr machen
  - Expressivität von **Assn**: schwächste Vorbedingung als Zusicherung $w[\![c, B]\!]$ ausdrückbar
- Beweis konstruiert $w[\![c, B]\!]$, aber nicht praktisch nutzbar.

Vorlesung vom 16.05.06: Generierung von Verifikationsbedingungen

Wo sind wir?

- Vorletzte Woche: Korrektheit des Hoare-Kalküls
- Letzte Woche: Vollständigkeit des Hoare-Kalküls
- Diese Woche: Verifikationsbedingungen

Ziel

- Ziel: maschinelle Verifikation von Code

  Eingabe: $\{A\}c\{B\}$

  Ausgabe: `True` wenn $\models \{A\}c\{B\}$
  `False` wenn $\neg \models \{A\}c\{B\}$

- Warum nicht $w[\![c, B]\!]$ nutzen?

$$\models \{A\}c\{B\} \Longleftrightarrow \models w[\![c, B]\!]$$

Ziel

- Ziel: maschinelle Verifikation von Code

  Eingabe: $\{A\}c\{B\}$

  Ausgabe: `True` wenn $\models \{A\}c\{B\}$
  `False` wenn $\neg \models \{A\}c\{B\}$

- Warum nicht $w[\![c, B]\!]$ nutzen?

$$\models \{A\}c\{B\} \Longleftrightarrow \models w[\![c, B]\!]$$

- Problem: nicht handhabbar.

Annotierte Programme

- Idee: Programme mit Zusicherungen annotieren
  - An der Schleife: Invariante

Annotierte Programme

- Idee: Programme mit Zusicherungen annotieren
  - An der Schleife: Invariante
- Annotierte Anweisungen ($\mathbf{Com}_A$):

$$
\begin{aligned}
c ::= \ & \texttt{skip} \mid x := a \mid \\
& c; (x := a) \mid c; \{D\} c_0 \\
& \texttt{if } b \texttt{ then } c \texttt{ else } c \texttt{ fi} \mid \texttt{while } b \texttt{ do } \{D\} c \texttt{ od}
\end{aligned}
$$

  mit c_0 keine Zuweisung
- Idee: Zusicherung ist an dieser Stelle immer wahr

Verifikationsbedingungen

$$vc : \textbf{Assn} \times \textbf{Com}_A \times \textbf{Assn} \to \mathbb{P}(\textbf{Assn})$$

$$\begin{aligned}
vc(\{A\}\texttt{skip}\{B\}) &= \{A \Longrightarrow B\} \\
vc(\{A\}X := a\{B\}) &= \{A \Longrightarrow B\,[a/X]\} \\
vc(\{A\}c; X := a\{B\}) &= vc(\{A\}c\{B\,[a/X]\}) \\
vc(\{A\}c; \{D\}c_0\{B\}) &= vc(\{A\}c\{D\}) \cup vc(\{D\}c_0\{B\}) \\
vc(\{A\}\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ fi}\{B\}) &= vc(\{A \wedge b\}c_1\{B\}) \\
&\quad \cup vc(\{A \wedge \neg b\}c_2\{B\}) \\
vc(\{A\}\texttt{while } b \texttt{ do } \{D\}c \texttt{ od}\{B\}) &= vc(\{D \wedge b\}c\{D\}) \\
&\quad \cup \{A \Longrightarrow D\} \\
&\quad \cup \{D \wedge \neg b \Longrightarrow B\}
\end{aligned}$$

Korrektheit der Verifikationsbedingungen

Satz (Korrektheit): Wenn für alle $D \in vc\{A\}c\{B\}$, $\models D$, dann $\models \{A\}c\{B\}$.

Beweis: Wir zeigen $\vdash \{A\}c\{B\}$ mit struktureller Induktion über $c \in \mathbf{Com}_A$.

- $c \equiv \texttt{skip}$
- $c \equiv X := a$
- $c \equiv c_0; \{D\}c_1$
- $c \equiv \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ fi}$
- $c \equiv \texttt{while } b \texttt{ do } \{D\}c \texttt{ od}$

Zusammenfassung

- Konstruktion von $w[\![c, B]\!]$ aus Vollständigkeitsnbeweis unbrauchbar.
- Daher: mit Zusicherungen annotierte Programme
- Daraus: Generierung von Verifikationsbedingungen
- Wenn alle Verifikationsbedingungen beweisbar, Programm korrekt.
- Aber: Verifikationsbedingungen können unbeweisbar sein, wenn
  - Programm falsch
  - Zusicherungen unerfüllbar
- Schwachpunkt: Grammatik

Vorlesung vom 23.05.06: Generierung von Verifikationsbedingungen: Beispiele

Wo sind wir?

- Vorletzte Wochen: Korrektheit, Vollständigkeit des Hoare-Kalküls
- Letzte Woche: Verifikationsbedingungen
- Diese Woche: Beispiele für Generierung von Verifikationsbedingungen

Beispiel 1: `swap`

$$\texttt{swap} \quad \equiv \quad t := x; x := y; y := t$$

Beispiel 1: `swap`

$$\begin{aligned} \texttt{swap} &\equiv t := x; x := y; y := t \\ pre &\equiv x = X \texttt{ and } y = Y \\ post &\equiv x = Y \texttt{ and } y = X \end{aligned}$$

Beispiel 2: max

$$\texttt{max} \quad \equiv \quad \texttt{if } x < y \texttt{ then } m := y \texttt{ else } m := x \texttt{ fi}$$

Beispiel 2: max

$$\texttt{max} \equiv \texttt{if } x \texttt{ < } y \texttt{ then } m := y \texttt{ else } m := x \texttt{ fi}$$

$$pre \equiv \texttt{True}$$

$$post \equiv x \texttt{ <= } m \texttt{ and } y \texttt{ <= } m$$

Beispiel 3: `sum`

$$\begin{aligned}\texttt{sum} \quad \equiv \quad & i := 0; s := 0; \\ & \{J\}\,\texttt{while not}\ (i = n)\ \texttt{do}\ \{I\} i := i + 1; s := s + i\ \texttt{od}\end{aligned}$$

Beispiel 3: `sum`

$$
\begin{aligned}
\texttt{sum} \quad &\equiv \quad i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while not}\ (i = n)\ \texttt{do}\ \{I\} i := i + 1; s := s + i\ \texttt{od} \\
\\
\mathit{pre} \quad &\equiv \quad 0 <= n \\
\\
\mathit{post} \quad &\equiv \quad s = \mathit{sum}(1, n)
\end{aligned}
$$

Beispiel 3: `sum`

$$
\begin{aligned}
\texttt{sum} &\equiv i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while not } (i = n)\ \texttt{do } \{I\} i := i + 1; s := s + i\ \texttt{od} \\
pre &\equiv 0 <= n \\
post &\equiv s = sum(1, n) \\
I &\equiv s = sum(1, i) \\
J &\equiv i = 0 \texttt{ and } s = 0
\end{aligned}
$$

Beispiel 3a: `sum`

$$\texttt{sum} \equiv i := 0; s := 0; \{J\}\,\texttt{while}\ i < n\ \texttt{do}\ \{I\} s := s + i; i := i + 1\ \texttt{od}$$

Beispiel 3a: `sum`

$$
\begin{aligned}
\texttt{sum} &\equiv i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while}\ i\ \texttt{<}\ n\ \texttt{do}\ \{I\} s := s\ \texttt{+}\ i; i := i\ \texttt{+}\ 1\ \texttt{od} \\
pre &\equiv 0\ \texttt{<=}\ n \\
post &\equiv s = sum(1, n)
\end{aligned}
$$

Beispiel 3a: `sum`

$$
\begin{aligned}
\texttt{sum} &\equiv i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while}\ i\ \texttt{<}\ n\ \texttt{do}\ \{I\} s := s\ \texttt{+}\ i; i := i\ \texttt{+}\ 1\ \texttt{od} \\
pre &\equiv 0\ \texttt{<=}\ n \\
post &\equiv s = sum(1, n) \\
I &\equiv s = sum(1, (i+1))\ \texttt{and}\ i\ \texttt{<=}\ n \\
J &\equiv i\ \texttt{=}\ 0\ \texttt{and}\ s\ \texttt{=}\ 0
\end{aligned}
$$

Beispiel 4: `times`

$$\texttt{times} \equiv \quad i := 0; p := 0;$$
$$\{J\}\,\texttt{while}\ i < n\ \texttt{do}\ \{I\}p := p + m; i := i + 1\ \texttt{od}$$

Beispiel 4: `times`

$$
\begin{aligned}
\texttt{times} &\equiv i := 0; p := 0; \\
&\quad \{J\}\,\texttt{while}\ i\ \texttt{<}\ n\ \texttt{do}\ \{I\} p := p\ \texttt{+}\ m; i := i\ \texttt{+}\ 1\ \texttt{od} \\
\mathit{pre} &\equiv 0\ \texttt{<=}\ n \\
\mathit{post} &\equiv p = m\ \texttt{*}\ n
\end{aligned}
$$

Beispiel 4: `times`

$$
\begin{aligned}
\texttt{times} &\equiv i := 0; p := 0; \\
&\quad \{J\}\,\texttt{while}\ i < n\ \texttt{do}\ \{I\} p := p + m; i := i + 1\ \texttt{od} \\
pre &\equiv 0 <= n \\
post &\equiv p = m * n \\
I &\equiv p = m * i\ \texttt{and}\ i <= n \\
J &\equiv i = 0\ \texttt{and}\ p = 0
\end{aligned}
$$

Beispiel 5: `div`

$$\texttt{div} \equiv q := 0; r := x; \{J\}\,\texttt{while not}\ (r\ \texttt{<}\ y)\ \texttt{do}\ \{I\} r := r\ \texttt{-}\ y; q := q\ \texttt{+}\ 1\ \texttt{od}$$

Beispiel 5: `div`

$$
\begin{aligned}
\texttt{div} \quad &\equiv \quad q := 0; r := x; \\
&\qquad \{J\}\,\texttt{while not}\ (r \texttt{ < } y)\ \texttt{do}\ \{I\} r := r \texttt{ - } y; q := q \texttt{ + } 1\ \texttt{od} \\
\mathit{pre} \quad &\equiv \quad 0 \texttt{ <= } x \ \texttt{and}\ 0 \texttt{ <= } y \\
\mathit{post} \quad &\equiv
\end{aligned}
$$

Beispiel 5: `div`

$$
\begin{aligned}
\texttt{div} \quad &\equiv \quad q := 0; r := x; \\
&\qquad \{J\}\,\texttt{while not}\ (r \texttt{ < } y)\ \texttt{do}\ \{I\} r := r \texttt{ - } y; q := q \texttt{ + } 1\ \texttt{od} \\
pre \quad &\equiv \quad 0 \texttt{ <= } x \texttt{ and } 0 \texttt{ <= } y \\
post \quad &\equiv \quad x \texttt{ = } q \texttt{ * } y \texttt{ + } r \texttt{ and } 0 \texttt{ <= } r \texttt{ and } r \texttt{ < } y \\
I \quad &\equiv \quad x \texttt{ = } q \texttt{ * } y \texttt{ + } r \texttt{ and } 0 \texttt{ <= } r \\
J \quad &\equiv \quad q \texttt{ = } 0 \texttt{ and } r \texttt{ = } x
\end{aligned}
$$

Beispiel 6: `euclid`

$$
\begin{array}{lll}
\texttt{euclid} & \equiv & a := x; b := y; \{J\} \\
& & \texttt{while not } (a = b) \texttt{ do} \\
& & \quad \{I\} \texttt{ if } a \texttt{ <= } b \texttt{ then } b := b - a \\
& & \qquad\qquad\qquad\quad \texttt{else } a := a - b \\
& & \qquad\qquad\qquad\quad \texttt{fi od}
\end{array}
$$

Beispiel 6: `euclid`

$$
\begin{aligned}
\texttt{euclid} \quad \equiv \quad & a := x; b := y; \{J\} \\
& \texttt{while not } (a = b) \texttt{ do} \\
& \quad \{I\} \texttt{ if } a \texttt{ <= } b \texttt{ then } b := b - a \\
& \qquad\qquad\qquad\quad \texttt{else } a := a - b \\
& \qquad\qquad\qquad\quad \texttt{fi od}
\end{aligned}
$$

$$
pre \quad \equiv \quad \texttt{True}
$$

$$
post \quad \equiv \quad a = \gcd(x, y)
$$

Beispiel 6: `euclid`

$$
\begin{aligned}
\texttt{euclid} \quad \equiv \quad & a := x; b := y; \{J\} \\
& \texttt{while not } (a = b) \texttt{ do} \\
& \quad \{I\} \texttt{ if } a \texttt{ <= } b \texttt{ then } b := b - a \\
& \qquad\qquad\qquad\quad \texttt{else } a := a - b \\
& \qquad\qquad\qquad\quad \texttt{fi od}
\end{aligned}
$$

$$pre \equiv \texttt{True}$$

$$post \equiv a = \gcd(x, y)$$

$$I \equiv \gcd(x, y) = \gcd(a, b)$$

$$J \equiv x \texttt{ = } a \texttt{ and } b \texttt{ = } y$$

Beispiel 7: pow

$$
\begin{array}{lll}
\texttt{pow} & \equiv & \texttt{while not } (y = 0) \texttt{ do} \\
& & \quad \{I_1\} \texttt{ while } even(y)) \texttt{ do} \\
& & \qquad \{I_2\}\, x := x * x; y := y/2; \\
& & \qquad \texttt{od}; \\
& & \quad z := z * x; y := y - 1 \\
& & \quad \texttt{od}
\end{array}
$$

Beispiel 7: pow

$$
\begin{array}{rcl}
\texttt{pow} & \equiv & \texttt{while not } (y = 0) \texttt{ do} \\
& & \quad \{I_1\} \texttt{ while } even(y)) \texttt{ do} \\
& & \quad\quad \{I_2\}\, x := x * x; y := y/2; \\
& & \quad\quad \texttt{od}; \\
& & \quad z := z * x; y := y - 1 \\
& & \quad \texttt{od} \\
\mathit{pre} & \equiv & x = m \texttt{ and } y = n \texttt{ and } z = 1 \\
\mathit{post} & \equiv & z = m^n
\end{array}
$$

Beispiel 7: pow

$$
\begin{aligned}
\texttt{pow} \quad &\equiv \quad \texttt{while not } (y = 0) \texttt{ do} \\
&\qquad \{I_1\} \texttt{ while } even(y)) \texttt{ do} \\
&\qquad\qquad \{I_2\}\, x := x * x; y := y/2; \\
&\qquad\qquad \texttt{od}; \\
&\qquad z := z * x; y := y - 1 \\
&\qquad \texttt{od} \\
pre \quad &\equiv \quad x = m \texttt{ and } y = n \texttt{ and } z = 1 \\
post \quad &\equiv \quad z = m^n \\
I_1 \quad &\equiv \quad m^n = z * x^y \\
I_2 \quad &\equiv \quad m^n = z * x^y
\end{aligned}
$$

Beispiel 8: `isqr`

$$
\begin{aligned}
\texttt{isqr} \quad \equiv \quad & t := 1; s := t; i := 0; \{J\} \\
& \texttt{while } s \texttt{ <= } a \texttt{ do } \{I\} \\
& \quad t := t + 2; \\
& \quad s := s + t; \\
& \quad i := i + 1 \texttt{fi od}
\end{aligned}
$$

Beispiel 8: `isqr`

$$
\begin{array}{rcl}
\texttt{isqr} & \equiv & t := 1; s := t; i := 0; \{J\} \\
& & \texttt{while } s \texttt{ <= } a \texttt{ do } \{I\} \\
& & \quad t := t + 2; \\
& & \quad s := s + t; \\
& & \quad i := i + 1 \texttt{fi od} \\
\mathit{pre} & \equiv & 0 \texttt{ <= } a \\
\mathit{post} & \equiv & i \texttt{ * } i \texttt{ <= } a \texttt{ and } a \texttt{ < } (i + 1) \texttt{ * } (i + 1)
\end{array}
$$

Beispiel 8: `isqr`

$$
\begin{aligned}
\texttt{isqr} \quad &\equiv \quad t := 1; s := t; i := 0; \{J\} \\
&\qquad \texttt{while } s \texttt{ <= } a \texttt{ do } \{I\} \\
&\qquad\quad t := t + 2; \\
&\qquad\quad s := s + t; \\
&\qquad\quad i := i + 1 \texttt{fi od} \\
\mathit{pre} \quad &\equiv \quad 0 \texttt{ <= } a \\
\mathit{post} \quad &\equiv \quad i \texttt{ * } i \texttt{ <= } a \texttt{ and } a \texttt{ < } (i+1) \texttt{ * } (i+1) \\
I \quad &\equiv \quad ? \\
J \quad &\equiv \quad t \texttt{ = } 1 \texttt{ and } s \texttt{ = } 1 \texttt{ and } i \texttt{ = } 0
\end{aligned}
$$

$$
\mathit{Hinweis}: \qquad 1 + 3 + 5 + \ldots + (2n - 1) = n^2
$$

Zusammenfassung

- Auch einfache Programme generieren umfangreiche Verifikationsbedingungen
- Finden der Invariante nicht immer einfach
- Rest mechanisch
- Deduktion (zur Vereinfachung) vonnöten

Vorlesung vom 30.05.06: Totale Korrektheit

Wo sind wir?

- Hoare-Kalkül:
  - Korrektheit und Vollständigkeit des Hoare-Kalküls für partielle Korrektheit
  - Generierung von Verifikationsbedingungen
  - Diese Woche: Totale Korrektheit
- Jenseits von Hoare: JML und WHY
- Werkzeugunterstützung
- Korrekte Programmentwicklung

Warum totale Korrektheit?

- Intutiver: Annahme Termination implizit bei Spezifikation
- Siehe Beispiel:

$$\vdash \{T\}\texttt{skip}\{T\}$$

$$\vdash \{T \wedge T\}\texttt{skip}\{T\}$$

$$\vdash \{T\}\texttt{while } T \texttt{ do skip od}\{T \wedge \neg T\}$$

Totale Korrektheit

- Totale Korrektheit: $[A]P[B]$ gilt gdw.

  Wenn Zustand $\sigma \models A$, dann:

  - P terminiert
  - Nach Ausführung gilt B

- Formal:

$$[A]c[B] = \forall I.\forall \sigma.(\sigma \models^I A \Longrightarrow \mathcal{C}[\![c]\!]\sigma \neq \bot \wedge \mathcal{C}[\![c]\!]\sigma \models^I B)$$

  - NB: Quantoren — c muß nicht immer terminieren.
  - Für Termination ist Interpretation fest aber beliebig.

Regeln für Totale Korrektheit

- Implikation ✓

Regeln für Totale Korrektheit

- Implikation ✓
- Skip ✓

Regeln für Totale Korrektheit

- Implikation ✓
- Skip ✓
- Zuweisungsregel ✓!

Regeln für Totale Korrektheit

- Implikation ✓
- Skip ✓
- Zuweisungsregel ✓!
- Sequenzregel ✓

Regeln für Totale Korrektheit

- Implikation ✓
- Skip ✓
- Zuweisungsregel ✓!
- Sequenzregel ✓
- Fallunterscheidung ✓

Regeln für Totale Korrektheit

- Implikation ✓
- Skip ✓
- Zuweisungsregel ✓!
- Sequenzregel ✓
- Fallunterscheidung ✓
- Schleifenregel ✘

Zuweisungen

$X := A$ Problem: Auswertung von A muss terminieren

- Rekursive Funktionen
  - Nicht in IMP — aber für Erweiterungen!

Zuweisungen

$X := A$ Problem: Auswertung von A muss terminieren

- Rekursive Funktionen
  - Nicht in IMP — aber für Erweiterungen!
- Fehlerfälle:
  - Division durch 0, Bereichsüberschreitung
  - Werden nicht durch $\bot$ modelliert — $\bot$ ist Nichttermination
  - Fehlerwerte (e.g. für Fließkommazahlen IEEE 754 und 854: `Inf`, `NaN`)
  - Ausnahmen (e.g. Haskell, Java: Division durch 0 für `Int`)
  - Modulo-Arithmetik (e.g. `Int` $\cong \mathbb{Z}_{2^N}$ mit Wortbreite N)

Schleifenregel

- Idee: wohlfundierte Ordnung — hier $\mathbb{N}$
  - Genauer positive ganze Zahlen
- Jede Iteration muss Variante erniedrigen

Schleifenregel

- Idee: wohlfundierte Ordnung — hier `N`
  - Genauer positive ganze Zahlen
- Jede Iteration muss Variante erniedrigen
- Formal:

$$\frac{\vdash [I \wedge b \wedge E = n]c[I \wedge E < n] \quad \vdash I \wedge b \Longrightarrow 0 \leq E}{\vdash [I]\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}[I \wedge \neg b]}\ While$$

$$\frac{\vdash [I \wedge b]c[I] \quad \vdash [I \wedge b \wedge E = n]c[E < n] \quad \vdash I \wedge b \Longrightarrow 0 \leq E}{\vdash [I]\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}[I \wedge \neg b]}\ While$$

  wobei n nicht in I, c, b, E auftritt.
- Andere Regeln bleiben

Hoare-Kalkül für totale Korrektheit

$$\frac{}{\vdash [A]\texttt{skip}[A]}\,Skip \qquad \frac{}{\vdash [B[a/X]]X := a[B]}\,Assgn$$

$$\frac{\vdash [A]c_0[C] \quad \vdash [C]c_1[B]}{\vdash [A]c_0; c_1[B]}\,Seq$$

$$\frac{\vdash [A \wedge b]c_0[B] \quad \vdash [A \wedge \neg b]c_1[B]}{\vdash [A]\texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \texttt{ fi}[B]}\,Cond$$

$$\frac{\vdash [I \wedge b]c[I] \quad \vdash [I \wedge b \wedge E = n]c[E < n] \quad \vdash I \wedge b \Longrightarrow 0 \leq E}{\vdash [I]\texttt{while } b \texttt{ do } c \texttt{ od}[I \wedge \neg b]}\,While$$

$$\frac{\vdash (A \Longrightarrow A') \quad \vdash [A']c[B'] \quad \vdash (B' \Longrightarrow B)}{\vdash [A]c[B]}\,Impl$$

Beispiel

- Sei $p \equiv$ `while` $Y \leq R$ `do` $R := R - Y;$
  $Q := Q + 1$
  `od`
- Zu zeigen: $\vdash [0 < Y]p[true]$

Beispiel

- Sei $p \equiv$ `while` $Y \leq R$ `do` $R := R - Y;$
  $Q := Q + 1$
  `od`
- Zu zeigen: $\vdash [0 < Y]p[true]$
- Schleifenregel
  - Invariante: $I \equiv 0 < Y$
  - Variante: $E \equiv R$

Verifikationsbedingungen für Totale Korrektheit

- Benötigt Annotation mit Varianten:

$$
\begin{array}{rl}
c ::= & \texttt{skip} \mid x := a \mid \\
& c; (x := a) \mid c; \{D\} c_0 \\
& \texttt{if } b \texttt{ then } c \texttt{ else } c \texttt{ fi} \mid \texttt{while } b \texttt{ do } \{D\}[E] c \texttt{ od}
\end{array}
$$

$$
\begin{aligned}
vc([A]\texttt{while } b \texttt{ do } \{D\}[E] c \texttt{ od}[B]) = \; & vc([D \wedge b \wedge E = n] c [D \wedge E < n]) \\
& \cup \{ D \wedge b \Longrightarrow 0 \leq E, \\
& \quad A \Longrightarrow D, \\
& \quad D \wedge \neg b \Longrightarrow B \}
\end{aligned}
$$

  wobei n nicht in A, b, D, c, B, E auftritt

- Andere Regeln bleiben.

Verifikationsbedingungen für Totale Korrektheit

$$vc : \textbf{Assn} \times \textbf{Com}_A \times \textbf{Assn} \to \mathbb{P}(\textbf{Assn})$$

$$\begin{aligned}
vc([A]\texttt{skip}[B]) &= \{A \Longrightarrow B\} \\
vc([A]X := a[B]) &= \{A \Longrightarrow B\,[a/X]\} \\
vc([A]c; X := a[B]) &= vc([A]c[B\,[a/X]]) \\
vc([A]c; \{D\}c_0[B]) &= vc([A]c[D]) \cup vc([D]c_0[B]) \\
vc([A]\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ fi}[B]) &= vc([A \wedge b]c_1[B]) \\
&\quad \cup vc([A \wedge \neg b]c_2[B]) \\
vc([A]\texttt{while } b \texttt{ do } \{D\}[E]c \texttt{ od}[B]) &= vc([D \wedge b]c[D]) \\
&\quad \cup vc([D \wedge b \wedge E = n]c[E < n]) \\
&\quad \cup \{D \wedge b \Longrightarrow 0 \leq E, \\
&\qquad A \Longrightarrow D, \\
&\qquad D \wedge \neg b \Longrightarrow B\}
\end{aligned}$$

Beispiel 3: `sum`

$$\texttt{sum} \equiv i := 0; s := 0; \{J\}\,\texttt{while not } (i = n)\ \texttt{do } \{I\}[E]i := i + 1; s := s + i\ \texttt{od}$$

Beispiel 3: `sum`

$$
\begin{aligned}
\texttt{sum} \quad &\equiv \quad i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while not}\ (i = n)\ \texttt{do}\ \{I\}[E] i := i + 1; s := s + i\ \texttt{od} \\
\mathit{pre} \quad &\equiv \quad 0 <= n \\
\mathit{post} \quad &\equiv \quad s = \mathit{sum}(1, n)
\end{aligned}
$$

Beispiel 3: `sum`

$$
\begin{aligned}
\texttt{sum} &\equiv i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while not}\ (i = n)\ \texttt{do}\ \{I\}[E] i := i + 1; s := s + i\ \texttt{od} \\
pre &\equiv 0 <= n \\
post &\equiv s = sum(1, n) \\
I &\equiv s = sum(1, i) \\
J &\equiv i = 0\ \texttt{and}\ s = 0 \\
E &\equiv n - i
\end{aligned}
$$

Beispiel 3: `sum`

$$
\begin{aligned}
\texttt{sum} &\equiv i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while not } (i = n)\ \texttt{do } \{I\}[E] i := i + 1; s := s + i\ \texttt{od} \\
pre &\equiv 0 <= n \\
post &\equiv s = sum(1, n) \\
I &\equiv s = sum(1, i) \\
J &\equiv i = 0 \texttt{ and } s = 0 \\
E &\equiv n - i
\end{aligned}
$$

Aber: Invariante zu schwach.

Beispiel 3a: `sum`

$$\texttt{sum} \equiv i := 0; s := 0; \{J\}\,\texttt{while}\ i < n\ \texttt{do}\ \{I\}[E]s := s + i; i := i + 1\ \texttt{od}$$

Beispiel 3a: `sum`

$$
\begin{aligned}
\texttt{sum} \quad &\equiv \quad i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while}\ i \texttt{ < } n\ \texttt{do}\ \{I\}[E]s := s \texttt{ + } i; i := i \texttt{ + } 1\ \texttt{od} \\
\mathit{pre} \quad &\equiv \quad 0 \texttt{ <= } n \\
\mathit{post} \quad &\equiv \quad s = \mathit{sum}(1, n)
\end{aligned}
$$

Beispiel 3a: sum

$$
\begin{aligned}
\texttt{sum} &\equiv i := 0; s := 0; \\
&\quad \{J\}\,\texttt{while}\ i < n\ \texttt{do}\ \{I\}[E] s := s + i; i := i + 1\ \texttt{od} \\
pre &\equiv 0 <= n \\
post &\equiv s = sum(1, n) \\
J &\equiv i = 0\ \texttt{and}\ s = 0 \\
I &\equiv s = sum(1, (i + 1))\ \texttt{and}\ i <= n \\
E &\equiv n - i
\end{aligned}
$$

Beispiel 4: `times`

$$\texttt{times} \equiv i := 0; p := 0;$$
$$\{J\}\,\texttt{while}\ i < n\ \texttt{do}\ \{I\}[E]p := p + m; i := i + 1\ \texttt{od}$$

Beispiel 4: `times`

$$
\begin{aligned}
\texttt{times} &\equiv i := 0; p := 0; \\
&\quad \{J\}\,\texttt{while}\ i \texttt{ < } n\ \texttt{do}\ \{I\}[E]p := p \texttt{ + } m; i := i \texttt{ + } 1\ \texttt{od} \\
pre &\equiv 0 \texttt{ <= } n \\
post &\equiv p = m \texttt{ * } n
\end{aligned}
$$

Beispiel 4: `times`

$$
\begin{aligned}
\texttt{times} &\equiv i := 0; p := 0; \\
&\quad \{J\}\,\texttt{while}\ i\ \texttt{<}\ n\ \texttt{do}\ \{I\}[E]p := p\ \texttt{+}\ m; i := i\ \texttt{+}\ 1\ \texttt{od} \\
pre &\equiv 0\ \texttt{<=}\ n \\
post &\equiv p = m\ \texttt{*}\ n \\
J &\equiv i\ \texttt{=}\ 0\ \texttt{and}\ p\ \texttt{=}\ 0 \\
I &\equiv p = m * i\ \texttt{and}\ i\ \texttt{<=}\ n \\
E &\equiv n - i
\end{aligned}
$$

Beispiel 5: `div`

$$\texttt{div} \equiv q := 0; r := x; \{J\}\,\texttt{while not}\ (r\ \texttt{<}\ y)\ \texttt{do}\ \{I\}[E]r := r\ \texttt{-}\ y; q := q\ \texttt{+}\ 1\ \texttt{od}$$

Beispiel 5: `div`

$$
\begin{aligned}
\texttt{div} \quad &\equiv \quad q := 0; r := x; \\
&\qquad \{J\}\,\texttt{while not}\ (r \texttt{ < } y)\ \texttt{do}\ \{I\}[E]r := r \texttt{ - } y; q := q \texttt{ + } 1\ \texttt{od} \\
\mathit{pre} \quad &\equiv \quad 0 \texttt{ <= } x \texttt{ and } 0 \texttt{ < } y \\
\mathit{post} \quad &\equiv \quad x \texttt{ = } q \texttt{ * } y \texttt{ + } r \texttt{ and } 0 \texttt{ <= } r \texttt{ and } r \texttt{ < } y
\end{aligned}
$$

Beispiel 5: `div`

$$
\begin{aligned}
\texttt{div} &\equiv\ q := 0; r := x; \\
&\quad \{J\}\,\texttt{while not}\ (r \texttt{ < } y)\ \texttt{do}\ \{I\}[E]r := r \texttt{ - } y; q := q \texttt{ + } 1\ \texttt{od} \\
pre &\equiv\ 0 \texttt{ <= } x \texttt{ and } 0 \texttt{ < } y \\
post &\equiv\ x \texttt{ = } q \texttt{ * } y \texttt{ + } r \texttt{ and } 0 \texttt{ <= } r \texttt{ and } r \texttt{ < } y \\
J &\equiv\ q \texttt{ = } 0 \texttt{ and } r \texttt{ = } x \\
I &\equiv\ x \texttt{ = } q \texttt{ * } y \texttt{ + } r \texttt{ and } 0 \texttt{ <= } r \texttt{ and } 0 \texttt{ < } y \\
E &\equiv\ r
\end{aligned}
$$

Beispiel 6: `euclid`

$$
\begin{array}{lll}
\texttt{euclid} & \equiv & a := x; b := y; \{J\} \\
& & \texttt{while not } (a = b) \texttt{ do} \\
& & \quad \{I\}[E] \texttt{ if } a \texttt{ <= } b \texttt{ then } b := b - a \\
& & \qquad\qquad\qquad\quad \texttt{else } a := a - b \\
& & \qquad\qquad\qquad\quad \texttt{fi od}
\end{array}
$$

Beispiel 6: `euclid`

$$
\begin{aligned}
\texttt{euclid} \quad \equiv \quad & a := x; b := y; \{J\} \\
& \texttt{while not } (a = b) \texttt{ do} \\
& \quad \{I\}[E] \texttt{ if } a \texttt{ <= } b \texttt{ then } b := b - a \\
& \qquad\qquad\qquad\qquad \texttt{else } a := a - b \\
& \qquad\qquad\qquad\qquad \texttt{fi od}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{pre} \quad &\equiv \quad 0 \texttt{ < } x \texttt{ and } 0 \texttt{ < } y \\
\mathit{post} \quad &\equiv \quad a = \gcd(x, y)
\end{aligned}
$$

Beispiel 6: `euclid`

$$
\begin{aligned}
\texttt{euclid} \quad \equiv \quad & a := x; b := y; \{J\} \\
& \texttt{while not } (a = b) \texttt{ do} \\
& \quad \{I\}[E] \texttt{ if } a \texttt{ <= } b \texttt{ then } b := b - a \\
& \qquad\qquad\qquad\qquad \texttt{else } a := a - b \\
& \qquad\qquad\qquad\qquad \texttt{fi od}
\end{aligned}
$$

$$
\begin{aligned}
pre \quad &\equiv \quad 0 \texttt{ < } x \texttt{ and } 0 \texttt{ < } y \\
post \quad &\equiv \quad a = \gcd(x, y) \\
J \quad &\equiv \quad x \texttt{ = } a \texttt{ and } b \texttt{ = } y \\
I \quad &\equiv \quad \gcd(x, y) = \gcd(a, b) \texttt{ and } 0 \texttt{ < } a \texttt{ and } 0 \texttt{ < } b \\
E \quad &\equiv \quad a + b
\end{aligned}
$$

Beispiel 7: pow

$$
\begin{array}{lll}
\texttt{pow} & \equiv & \texttt{while not } (y = 0) \texttt{ do} \\
& & \quad \{I_1\}[E_1] \texttt{ while } even(y)) \texttt{ do} \\
& & \quad\quad \{I_2\}[E_2]\, x := x \texttt{ * } x; y := y/2; \\
& & \quad\quad \texttt{od}; \\
& & \quad z := z \texttt{ * } x; y := y - 1 \\
& & \quad \texttt{od}
\end{array}
$$

Beispiel 7: pow

$$
\begin{array}{rcl}
\texttt{pow} & \equiv & \texttt{while not } (y = 0) \texttt{ do} \\
& & \quad \{I_1\}[E_1] \texttt{ while } even(y)) \texttt{ do} \\
& & \quad\quad \{I_2\}[E_2]\, x := x \texttt{ * } x; y := y/2; \\
& & \quad\quad \texttt{od}; \\
& & \quad z := z \texttt{ * } x; y := y - 1 \\
& & \quad \texttt{od} \\
\mathit{pre} & \equiv & x = m \texttt{ and } y = n \texttt{ and } z = 1 \\
\mathit{post} & \equiv & z = m^n
\end{array}
$$

Beispiel 7: pow

$$
\begin{array}{rcl}
\texttt{pow} & \equiv & \texttt{while not } (y = 0) \texttt{ do} \\
& & \quad \{I_1\}[E_1] \texttt{ while } even(y)) \texttt{ do} \\
& & \quad\quad \{I_2\}[E_2]\, x := x \texttt{ * } x; y := y/2; \\
& & \quad\quad \texttt{od}; \\
& & \quad z := z \texttt{ * } x; y := y - 1 \\
& & \quad \texttt{od} \\
pre & \equiv & x = m \texttt{ and } y = n \texttt{ and } z = 1 \\
post & \equiv & z = m^n \\
I_1 & \equiv & m^n = z * x^y \\
I_2 & \equiv & m^n = z * x^y \\
E_1 & \equiv & y(?) \\
E_2 & \equiv & y(?)
\end{array}
$$

Beispiel 8: `isqr`

$$
\begin{aligned}
\texttt{isqr} \quad \equiv \quad & t := 1; s := t; i := 0; \{J\} \\
& \texttt{while } s \texttt{ <= } a \texttt{ do } \{I\}[E] \\
& \quad t := t + 2; \\
& \quad s := s + t; \\
& \quad i := i + 1 \texttt{fi od}
\end{aligned}
$$

Beispiel 8: `isqr`

$$
\begin{array}{rcl}
\texttt{isqr} & \equiv & t := 1; s := t; i := 0; \{J\} \\
& & \texttt{while } s \texttt{ <= } a \texttt{ do } \{I\}[E] \\
& & \quad t := t + 2; \\
& & \quad s := s + t; \\
& & \quad i := i + 1 \texttt{fi od} \\
\mathit{pre} & \equiv & 0 \texttt{ <= } a \\
\mathit{post} & \equiv & i \texttt{ * } i \texttt{ <= } a \texttt{ and } a \texttt{ < } (i + 1) \texttt{ * } (i + 1)
\end{array}
$$

Beispiel 8: `isqr`

$$
\begin{array}{rcl}
\texttt{isqr} & \equiv & t := 1; s := t; i := 0; \{J\} \\
 & & \texttt{while } s \texttt{ <= } a \texttt{ do } \{I\}[E] \\
 & & \quad t := t + 2; \\
 & & \quad s := s + t; \\
 & & \quad i := i + 1 \texttt{fi od} \\
pre & \equiv & 0 \texttt{ <= } a \\
post & \equiv & i \texttt{ * } i \texttt{ <= } a \texttt{ and } a \texttt{ < } (i + 1) \texttt{ * } (i + 1) \\
J & \equiv & t \texttt{ = } 1 \texttt{ and } s \texttt{ = } 1 \texttt{ and } i \texttt{ = } 0 \\
I & \equiv & ? \\
E & \equiv & a - s(?) \\
Hinweis: & & 1 + 3 + 5 + \ldots + (2n - 1) = n^2
\end{array}
$$

Zusammenfassung

- Totale Korrektheit = Partielle Korrektheit plus Termination

$$[A]c[B] = \{A\}c\{B\} \wedge [A]c[true]$$

- Kalkül für totale Korrektheit: Änderung an Schleifenregel
  - `while` erlaubt Nichttermination
  - Variante garantiert Termination
- Einfache Verallgemeinerung von Kalkül und Verifikationsbedingungen
- Nächste Wochen:
  Jenseits des Hoare-Kalküls — die Java Modelling Language (JML)

Vorlesung vom 06.06.06: The Java Modelling Language

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare:
  - Einführung
  - Reasoning in JML
  - Werkzeugunterstützung
- Korrekte Programmentwicklung

Was ist JML?

The Java Modeling Language (JML) is a behavioral interface specification language that can be used to specify the behavior of Java modules. It combines the design by contract approach of Eiffel and the model-based specification approach of the Larch family of interface specification languages, with some elements of the refinement calculus.

Von `http://www.jmlspecs.org/`

Was ist JML?

- JML spezifiziert Verhalten

Was ist JML?

- JML spezifiziert Verhalten
- Baut auf Java auf:
  - Strukturierung durch Java (Klassen)
  - Syntaktisch: Annotation von Programmen als Kommentar

Was ist JML?

- JML spezifiziert Verhalten
- Baut auf Java auf:
  - Strukturierung durch Java (Klassen)
  - Syntaktisch: Annotation von Programmen als Kommentar
- Methodologie: Design by Contract

Was ist JML?

- JML spezifiziert Verhalten
- Baut auf Java auf:
  - Strukturierung durch Java (Klassen)
  - Syntaktisch: Annotation von Programmen als Kommentar
- Methodologie: Design by Contract
- Sprachmittel:
  - Vor- und Nachbedingungen, Invarianten
  - Ausnahmeverhalten (exceptional behaviour)

Was ist JML?

- JML spezifiziert Verhalten
- Baut auf Java auf:
  - Strukturierung durch Java (Klassen)
  - Syntaktisch: Annotation von Programmen als Kommentar
- Methodologie: Design by Contract
- Sprachmittel:
  - Vor- und Nachbedingungen, Invarianten
  - Ausnahmeverhalten (exceptional behaviour)

Design-by-Contract

- Spezifikation definiert Vertrag zwischen
  - den Benutzern der Klasse
  - und den Implementieren
- Benutzer der Klasse
  - müssen Vorbedingung garantieren
  - dürfen auf Nachbedingung vertrauen
- Implementierer der Klassse
  - müssen Nachbedingung garantieren
  - dürfen Vorbedingung annehmen
- JML-Spezfikation macht Vertrag explizit

Vor- und Nachbedingungen

```
/*@ requires amount >= 0;
    ensures balance == \old(balance- amount) &&
            \result == balance; @*/
int debit(int amount)
{ ... }
```

- `\old` für vorherigen Wert
- `\result` für Resultat
- Beliebige Boolsche Ausdrücke

Ausnahmeverhalten (Exceptional Behaviour)

```
/* @ requires amount >= 0;
     ensures true;
     signals (BankException e)
             amount > balance &&
             balance == \old(balance) &&
             e.getReason().equals("Insufficient funds")  @*/
int debit(int amount) throws BankException;
```

- Spezifiziert Nachbedingung bei Ausnahme

  `signals (Exception) (Condition)`

- Abwesenheit von Ausnahmen:

  ```
  signals (Exception) false
  normal_behaviour
  ```

Klasseninvarianten

```
public class Wallet {
  public static final short MAX_BAL = 1000;
  private short balance;
  /*@ invariant 0 <= balance &&
                balance <= MAX_BAL;
    @*/
```

- Alle Methoden müssen Invariante erhalten
  - Invariante wird implizit Teil der Vor/Nachbedingung aller Methoden
- . . . auch unter Ausnahmen!

Klasseninvarianten

- Invarianten dokumentieren Entwurfsentscheidungen:

```
public class Directory {
 private File[] files;
 /*@ invariant files != null &&
     (\forall int i; 0<= i && i< files.length;
                     files[i] != null &&
                     files[i].getParent() == this);
  @*/
```

non_null

- Nützliche Abkürzung: `non_null`
- Verwendung in Vor/Nachbedingung oder Invariante

```
public class Directory {
  private /*@ non_null @*/File[] files;

  void createSubdir(/*@ non_null @*/String name) {...}
  ...
  Directory /*@ non_null @*/ getParent(){...}
```

Zusicherungen

- `assert` sichert Bedingung zu:

```
if (i<= 0 || j < 0) { ...
  } else if (j< 5) {
   //@ assert i>0 && 0<j && j< 5;
   ...
  } else {
   //@ assert i>0 && j> 5
   ...
  }
```

Zusicherungen

- `assert` Java-Schlüsselwort (seit Java1.4)
- JML: ausdrucksstärker, muss nicht ausführbar sein:

```
...
for (n= 0; n< a.length();n ++)
 if (a[n] == null) break;
 /@* assert (\forall int i; 0<= i && i< n; a[i] != null);
  @*/
```

Frame properties

- Frame properties schränken Seiteneffekte ein:

```
/*@ requires amount >= 0;
    assignable balance
    ensures balance == \old(balance- amount) &&
            \result == balance; @*/
int debit(int amount)
{ ... }
```

  - `debit` kann nur `balance` ändern

- Default: `assignable \everything`

Die Reine Lehre

- Methode ohne Seiteneffekte ist `pure`

```
public /*@ pure @*/ int getBalance(){...}

Directory /*@ pure non_null @*/getParent()...}
```

- Reine Methoden: implizit `assignable \nothing`
- (Ausschließlich) reine Methoden in Spezifikation verwendbar

```
/*@ invariant getBalance() && getBalance()<= MAX_BAL @*/
```

Noch ein Beispiel

```
import org.jmlspecs.models.JMLDouble;
public class SqrtExample {

 public final static double eps = 0.0001;

 //@ requires x >= 0.0;
 //@ ensures JMLDouble.approximatelyEqualTo(x, \result * \resu
 public static double sqrt(double x) {
     return Math.sqrt(x);
  }
}
```

Ein längeres Beispiel: `PriorityQueues`

Spezifikation in drei Schritten:

- Signatur: `PriorityQueueUser.java-refined`
- Verhaltensspezifikation: `PriorityQueueUser.jml-refined`
- Entwurfsspezifikation `PriorityQueueUser.java`

Ein längeres Beispiel:

PriorityQueueUser.java-refined

```
public interface PriorityQueueUser {
 /*@ pure @*/ boolean contains(Object argObj);
 /*@ pure @*/ Object next() throws PQException;
 void remove(Object argObj);
 /*@ pure @*/ boolean isEmpty();
}
```

Ein längeres Beispiel: PriorityQueueUser.jml-refined

```
//@ refine "PriorityQueueUser.java-refined";
//@ model import org.jmlspecs.models.*;

public interface PriorityQueueUser {
```

Ein längeres Beispiel: PriorityQueueUser.jml-refined

```
/*@ public model instance JMLValueSet entries;
  @ public initially entries != null && entries.isEmpty();
  @*/
/*@ public instance invariant entries != null
  @    && (\forall JMLType e; entries.has(e);
  @                           e instanceof QueueEntry);
  @ public instance invariant
  @   (\forall QueueEntry e1; entries.has(e1);
  @       (\forall QueueEntry e2;
  @         entries.has(e2) && !(e1.equals(e2));
  @           e2.obj != e1.obj
  @           && e2.timeStamp != e1.timeStamp ) );
  @*/ }
```

Ein längeres Beispiel: `PriorityQueues`

Spezifikation in drei Schritten:

- Signatur: `PriorityQueueUser.java-refined`
- Verhaltensspezifikation: `PriorityQueueUser.jml-refined`
- Entwurfsspezifikation: `PriorityQueueUser.java`
- Implementation: `PriorityQueue.java`
- `QueueEntry.jml-refined`, `QueueEntry.java`.

Ein längeres Beispiel: PriorityQueueUser.java

```
  package org.jmlspecs.samples.jmlkluwer;
//@ refine "PriorityQueueUser.jml-refined";

public interface PriorityQueueUser {

 /*@ also
   @   public normal_behavior
   @     ensures \result <==>
   @           argObj != null
   @           && (\exists QueueEntry e; entries.has(e);
   @                       e.obj == argObj);
   @*/
 /*@ pure @*/ boolean contains(Object argObj);
```

```
/*@ also
  @   public normal_behavior
  @     requires !entries.isEmpty();
  @     ensures
  @       (\exists QueueEntry r;
  @           entries.has(r) && \result == r.obj;
  @           (\forall QueueEntry o;
  @               entries.has(o) && !(r.equals(o));
  @               r.priorityLevel >= o.priorityLevel
  @               && (r.priorityLevel == o.priorityLevel
  @                   ==> r.timeStamp < o.timeStamp) ) );
```

```
  @ also
  @   public exceptional_behavior
  @     requires entries.isEmpty();
  @     signals_only PQException;
  @*/
/*@ pure @*/ Object next() throws PQException;
```

```
/*@ also
  @   public normal_behavior
  @     requires argObj != null && contains(argObj);
  @     assignable entries;
  @     ensures (\exists QueueEntry e;
  @         \old(entries.has(e)) && e.obj == argObj;
  @         entries.equals(\old(entries.remove(e))));
  @ also
  @   public normal_behavior
  @     requires argObj == null || !contains(argObj);
  @     assignable \nothing;
  @     ensures \not_modified(entries);
  @*/
```

```
 void remove(Object argObj);

 /*@ also
   @   public normal_behavior
   @     ensures \result <==> entries.isEmpty();
   @*/
 /*@ pure @*/ boolean isEmpty();
}
```

Zusammenfassung: JML

- Verhaltensorientierte Spezifikationssprache
- Java-Erweiterung: Annotierung von Java-Programmen
  - Vor/Nachbedingungen mit `requires` und `ensures`
  - Ausnahmeverhalten mit `signals`
  - Klasseninvarianten mit `invariant`
  - Zusicherungen mit `assert`
  - Seiteneffekte mit `assignable` oder `pure`
- Nächste Woche: Semantik von JML, Beweisen mit JML, Werkzeuge

Vorlesung vom 13.06.06: Ausnahmebehandlung (in Java und IMP)

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare:
  - Einführung
  - Reasoning in JML — Ausnahmen
  - Werkzeugunterstützung
- Korrekte Programmentwicklung

Eine Semantik für JML

- Problem: Semantik für JML braucht Semantik für Java.
- Verschiedene Ansätze:
  - Koalgebraisch (Huisman, Jacobs et al.)
  - Denotational (Leino et al.)
  - Axiomatisch (Poetzsch-Hefter; KeY)
- Plan heute:
  - Behandlung von Ausnahmen in **IMP**
  - Verallgemeinerung für Java: abrupte Termination

Ausnahmen

- Erweiterung von **IMP** um Ausnahmen: **Com**

$$c ::= \ldots \mid \texttt{throw}\ a \mid \texttt{try}\ c_0\ \texttt{catch}\ X.c_1$$

  Menge **Exn** von Ausnahmeausdrücken

- Semantik: Menge E von Ausnahmen

$$\mathcal{D} : \textbf{Com} \to (\Sigma \to (E * \Sigma) + \Sigma_\perp)$$
$$\mathcal{E}x : \textbf{Exn} \to (\Sigma \to E)$$

  - Entweder normaler Folgezustand $(inr(\sigma))$
  - . . . oder Ausnahme und Folgezustand $(inl(e, \sigma))$

- Auch Ausnahmen haben Folgezustand!

Semantik mit Ausnahmen

$$
\begin{aligned}
\mathcal{C}[\![\texttt{throw}\ a]\!] &\stackrel{def}{=} \lambda\sigma.inl(\mathcal{E}x[\![a]\!]\sigma, \sigma) \\
\mathcal{C}[\![c_0; c_1]\!] &\stackrel{def}{=} \lambda\sigma.cases\ \mathcal{C}[\![c_0]\!]\sigma\ \ of \\
&\qquad inr(\sigma') \to \mathcal{C}[\![\sigma_1]\!]\sigma' \\
&\qquad inl(e, \sigma') \to inl(e, \sigma') \\
\mathcal{C}[\![\texttt{try}\ c_0\ \texttt{catch}\ X.c_1]\!] &\stackrel{def}{=} \lambda\sigma.cases\ \mathcal{C}[\![c_0]\!]\sigma\ \ of \\
&\qquad inr(\sigma') \to inr(\sigma') \\
&\qquad inl(e, \sigma') \to \mathcal{C}[\![c_1]\!](\sigma'[\mathcal{E}x[\![a]\!]\sigma/X])
\end{aligned}
$$

Entsprechende Änderungen auch in $\mathcal{E}, \mathcal{A}$ (Ausnahmen in Ausdrücken!)

Hoare-Kalkül mit Ausnahmen

- Hoare-Tripel mit Ausnahmeverhalten
- $\{A\}c\{B \parallel X.S\}$
  - B Normale Nachbedingung
  - S Ausnahemverhalten, X kann Wert der Ausnahme binden
- Erfüllbarkeit:

$$\sigma \models^I \{A\}c\{B \parallel X.S\} \iff$$
$$\sigma \models^I A \Longrightarrow \mathit{cases}\, \mathcal{C}[\![c]\!]\sigma\ \mathit{of}\ \begin{array}{l} \mathit{inr}(\sigma') \to (\sigma' \models^I B) \\ \mathit{inl}(e, \sigma') \to (\sigma'[e/X] \models^I S) \end{array}$$

Hoare-Kalkül mit Ausnahmen

- Anpassung einiger Regeln:

$$\frac{\{A\}c_0\{C \parallel X.S\} \quad \{C\}c_1\{B \parallel X.S\}}{\{A\}c_0; c_1\{B \parallel X.S\}}\, Seq$$

- Neue Regeln:

$$\frac{}{\{A\}\texttt{throw}\ a\{\bot \parallel X.X = a \wedge A\}}\, Throw$$

wobei X nicht in A auftritt.

$$\frac{\{A\}c_0\{B \parallel X.S\} \quad \{A\}c_1\{S[X/a] \parallel T\}}{\{A\}\texttt{try}\ c_0\ \texttt{catch}\ X.c_1\{B \vee S[X/a] \parallel T\}}\, Catch$$

Ausnahmen in Java

- Java nutzt Ausnahmen für
  - normale Ausnahmen (`Exception`, `try`, `catch`)
  - Rückgabewerte (`return`)
  - Abbruch einer bestimmten Schleife (`break`)
  - Nichtabbruch einer bestimmten Iteration (`continue`)
- Viele verschiedene Fälle in Regeln!

Eine Java-Semantik

- Für **IMP**: Zustandstransformer $\Sigma \to \Sigma_\perp$ mit $\Sigma_\perp = \Sigma + 1$. Hier:

$$\begin{aligned} & \Sigma \to \Sigma_\perp + \mathbf{StatAbn} \\ \mathbf{StatAbn} &= \mathbf{Exn} + \mathbf{Return} + \mathbf{Break} + \mathbf{Cont} \\ \mathbf{Return} &= \Sigma * F \\ \mathbf{Break} &= \Sigma * L \\ \mathbf{Cont} &= \Sigma * L \end{aligned}$$

  mit F Menge von Funktionsnamen, L Menge von Labeln

- Für Ausdrücke vom Typ α:

$$\Sigma \to (\Sigma * \alpha)_\perp + \mathbf{Exn}$$

Java-Semantik: Weitere Feinheiten

- Speicherverwaltung in der JVM
  - Felder (Arrays)
  - Referenzen (z.B. auf Ausnahmen)
- Dynamische Bindung: Methodenaufruf
- Klassendefinitionen

Lokale Variablen

- Erweiterung von **Com** um Blöcke:

$$c ::= \ldots \mid \texttt{begin var } v_1, \ldots v_n; \ c \texttt{ end}$$

- Neue Regel:

$$\frac{\{A\}c\{B\}}{\{A\}\texttt{begin var } v_1, \ldots v_n; \ c \texttt{ end}\{B\}}\ Block$$

wobei $v_1, \ldots, v_n$ nicht in A oder B auftreten.

Zusammenfassung

- Java-Semantik (Ausblick)
- Besonderheiten von Java: Ausnahmeverhalten, lokale Variablen
- Erweiterung von **IMP** um einfache Ausnahmen
  - . . . aber Java hat viele Ausnahmen
- Blockregel für lokale Variablen

Vorlesung vom 20.06.06: Werkzeugunterstützung für JML

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare:
  - Einführung
  - Reasoning in JML — Ausnahmen
  - Werkzeugunterstützung
- Korrekte Programmentwicklung

Werkzeuge für JML

- Zur Laufzeit:
  - Prüfung von Zusicherungen
  - Generierung von Testfällen
- Zur Übersetzungszeit:
  - Statische Überprüfung,
  - Animation,
  - Theorembeweisen

Der JML-Übersetzer `jmlc`

- Übersetzt JML-Programm in Bytecode (spezielle JVM)
- Zur Laufzeit Überprüfung der Zusicherungen, Invarianten, Vor/-Nachbedingungen
- Verletzte Bedinungen: Fehlermeldung
- Nur für ausführbare Untermenge von JML
  - Konservativ (meldet nur echte Fehler)
- Erzeugt Fehlermeldung und backtrace

Unit Testing: `jmlunit` = JML + JUnit

- `jmlunit` generiert Unit Tests aus JML
- Tests prüfen Zusicherungen, Vor-/Nachbedingungen, Invarianten für Methode
- Beispiel:
  - `Purse.java`
  - `jmlunit` findet Ausnahme in `debit`: Ausgabe
  - Verstärkte Vorbedingung behebt Fehler:
    `amount >= 0 && amount <= balance`

Unit Testing: `jmlunit` = JML + JUnit

- Vorteile:
  - Prüft auch Spezifikation
  - Abdeckung der Zusicherungen (dazu nötig: vollständige Spezifikation)
- Nachteile:
  - Kann nur Fehler in einzelnen Methoden finden
  - Testdaten müssen gut ausgewählt werden
  - Testdaten werden nicht aus Spezifikation generiert

Extended Static Checking: ESC/Java

- Vollautomatische Programmverifikation:
  - Einfache Zusicherungen
  - Dereferenzierung von `null`
  - Feldzugriff außerhalb der Indexgrenze
  - Cast auf unzulässigen Typ

Extended Static Checking: ESC/Java

- Vollautomatische Programmverifikation:
  - Einfache Zusicherungen
  - Dereferenzierung von `null`
  - Feldzugriff außerhalb der Indexgrenze
  - Cast auf unzulässigen Typ
- Weder vollständig noch konsistent:
  - Findet nicht alle Fehler
  - Meldet nicht-existente Fehler
  - Pragmatik: es findet genug richtige Fehler!
- Benutzt externen, automatischen Beweiser
- Beispiel: Ausgabe

Exkurs: Abstrakte Interpretation

- Analyse durch Interpretation in einfacherem Wertebereich
- Beispiel:
  - $\mathbb{Z}$ interpretiert als $\mathbb{Z}^\# \stackrel{def}{=} \{plus, zerominus\}$
  - Abstrakte Version $*^\#$ der Multiplikation $*$

Exkurs: Abstrakte Interpretation

- Analyse durch Interpretation in einfacherem Wertebereich
- Beispiel:
  - $\mathbb{Z}$ interpretiert als $\mathbb{Z}^\# \stackrel{def}{=} \{plus, zerominus\}$
  - Abstrakte Version $*^\#$ der Multiplikation $*$
- Sicherheitskriterium:

$$\begin{array}{ccc} D & \xrightarrow{\quad f \quad} & D \\ {\scriptstyle abs}\downarrow & \supseteq & \downarrow{\scriptstyle abs} \\ D^\# & \xrightarrow[\quad f^\# \quad]{} & D^\# \end{array}$$

- Anwendung: Striktheitsanalyse

LOOP

- Echtes Theorembeweisen
- Coalgebraische Semantik (siehe letzte Woche)
- Flache Einbettung von Java in HOL
  - Garantierte Korrektheit
- LOOP:
  - Liest JML (`Java`)
  - Erzeugt Beweisverpflichtungen in Isabelle/PVS
- Nicht frei verfügbar — Forschungsprototyp

JACK

- Schwächste Vorbedingungen (wp) in Java für Java
- Implementation: Eclipse plug-in
- Ziel: specification for the working programmer
- Keine Formalisierung des wp-Kalküls, aber soll vollständig und konsistent sein
- Erzeugt Beweisverpflichtungen für automatische Theorembeweiser
  - Atelier B, Simplify, PVS
- Nicht frei verfügbar — lizenziert

WHY: multi-prover program verification

- `WHY`: schwächste Vorbedingung
- Sprach- und Beweiserunabhänging
- Liest annotierten Quelltext, produziert Beweisverpflichtungen
- Eingabesprachen: C, ML
- Beweiser: Coq, PVS, Isabelle, HOL Light, Mizar, Simplify, haRVey
- Zusätzliches Werkzeug für Java: Krakatoa
  - Liest JML, erzeugt Eingabe für `model` und Eingabe für WHY
- Frei verfügbar (GPL)

Zusammenfassung

- `jmlc`: Prüfung von Zusicherungen zur Laufzeit
- `jmlunit`: Generierung von Testfällen
- `ESC/Java2`: Statische Überprüfung; vollautomatisch, weder vollständig noch konsistent
- `LOOP`: Flache Einbettung der Semantik, Isabelle und PVS als Beweiser, interaktiv
- `JACK`: wp-Kalkül, verschiedene Beweiser (vollautomatisch)
- `WHY`: verschiedene Eingabesprachen (C, ML, JML `Krakatoa`), verschiedene Beweiser, interaktiv
- Altes Dilemma: Automatisierung vs. Mächtigkeit
- Nächste Woche: wie funktioniert `WHY`?

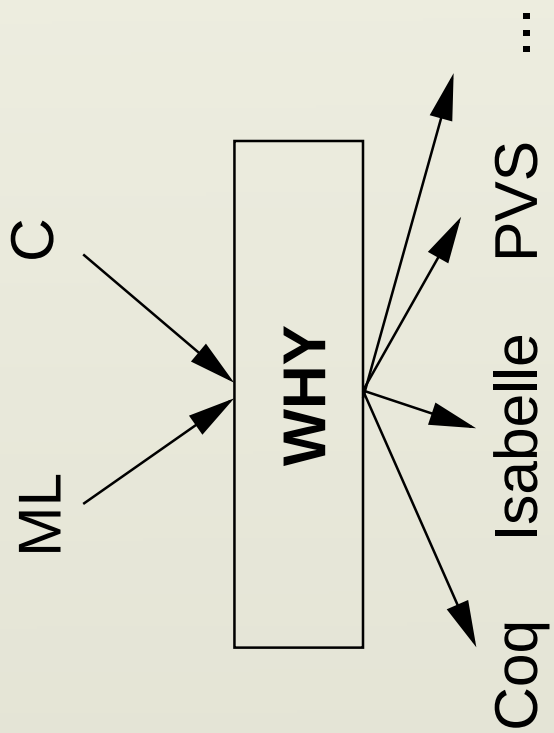
Vorlesung vom 27.06.06: Werkzeugunterstützte Verifikation

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare:
  - Einführung
  - Reasoning in JML — Ausnahmen
  - Werkzeugunterstützung
- Korrekte Programmentwicklung

WHY

- WHY: a Multi-Language Multi-Prover Verification Condition Generator



Wie funktioniert das?

- Übersetzung von ML/C/Java in interne Sprache WL
  - WL: Ausdrücke, Kommandos, lokale Variablen, Funktionen; Referenzen und Felder; Ausnahmen.
- Verifikations- und Vorbedingungskalkül für WL
- Theoretische Grundlage: Theorie der Effekte
- Funktionsweise: Generierung von Beweisverpflichtungen aus annotierten Programmen durch Interpretation

WL Konkrete Syntax: Terme und Programme

- Terme:

  $t ::= constant \mid x \mid x@L \mid f(t_1, \ldots, t_n)$

- Prädikate:

  $p ::= x \mid x(t_1, \ldots, t_n) \mid \texttt{true} \mid \texttt{false} \mid \texttt{not}\ p \mid$
  $p\ \texttt{and}\ p \mid p\ \texttt{or}\ p \mid \texttt{if}\ t\ \texttt{then}\ p\ \texttt{else}\ p \mid$
  $\texttt{forall}\ x : \beta.p \mid \texttt{exists}\ x : \beta.p$

- Programme:

  $e ::= \{p\}e\{p\} \mid t \mid !x \mid x := e \mid \texttt{ref}\ e$
  $x[e] \mid x[e] := e \mid e_1; e_2 \mid L : e \mid$
  $\texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid$
  $\texttt{fun}(x : \tau)\texttt{->}e \mid (ee) \mid \texttt{rec}\ x : \tau\{\texttt{variant}\ t\} = e \mid$
  $\texttt{while}\ e\ \texttt{do}\ \{\texttt{invariant}\ p\ \texttt{variant}\ t\}\ e\ \texttt{done}$

WL Konkrete Syntax: Effekte

- Effekte:

$$
\begin{array}{rcl}
\beta & ::= & \texttt{unit} \mid \texttt{bool} \mid \texttt{int} \mid \texttt{float} \\
\tau & ::= & \beta \mid \beta\ \texttt{ref} \mid \beta\ \texttt{array} \mid x : \tau \texttt{->} \kappa \\
\kappa & ::= & \{p\}\tau\epsilon\{p\} \\
\epsilon & ::= & \texttt{reads}\ x, \ldots, x\ \texttt{writes}\ x, \ldots, x
\end{array}
$$

- Effekte formalisieren Seiteneffekte
  - Typen und Terme wechselseitig rekursiv: abhängige Typen (dependent types)
  - Effekt enthält Vor-, Nachbedingung und Referenzen
  - Einschränkung: kein Aliasing für Referenzen

Vorbedingungen

- Für annotierte Tupel:

  $wp(\{p'\}e\{q'\}, q) = p' \wedge \forall result.\forall \omega.q' \Longrightarrow q$

- Für Ausdrücke (Auszug):

$$
\begin{aligned}
wp(t, q) &= q\,[t/result] \\
wp(!x, q) &= q\,[t/x] \\
wp(x := e, q) &= wp(e, q\,[\texttt{void}/result]\,[result/x] \\
wp(e_1; e_2) &= wp(e_1, wp(e_2, q)) \\
wp(L : e, q) &= wp(e, q)\,[x/x@L] \\
wp(\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3, q) & \\
&= wp(e_1, \texttt{if } result \texttt{ then } wp(e_2, q) \texttt{ else } wp(e_3, q)) \\
wp(\texttt{let } x = e_1 \texttt{ in } e_2, q) & \\
&= wp(e_1, wp(e_2, q)\,[result/x])
\end{aligned}
$$

Erweiterung für Ausnahmen

- Effekte und Nachbedingungen:

$$\epsilon \;::=\; \texttt{reads}\; x,\ldots,x\; \texttt{writes}\; x,\ldots,x\; \texttt{raises}\; E$$
$$\kappa \;::=\; \{p\}\tau\epsilon\{p \mid E \implies p\}$$

- Kommandos für Ausnahmen (und Hoare-Tripel)

$$e ::= \ldots \mid \texttt{raise}\ (Ee) \mid \texttt{try}\ e\ \texttt{with}\ Ex\texttt{->}e\ \texttt{end}$$

- Nachbedingungen: $wp(e,q,r)$ (Ausnahmenachbedingung r)

$$wp(\texttt{raise}\ (Ee),q,r) = wp(e,r,r)$$
$$wp(\texttt{try}\ e_1\ \texttt{with}\ Ex\texttt{->}e_2\ \texttt{end},q,r) = wp(e_1,q,wp(e_2,q,r)\,[result/x])$$

Verifikation von C-Programmen

- Übersetzung von C nach WL, dann Verifikation.
- Zwei Wege:
- High-level-Übersetzung:
  - Einfachere Beweisverpflichtungen
  - Nicht den vollen C-Sprachumfang
- Low-level-Übersetzung (Caduceus):
  - Kompliziertere Beweisverpflichtungen
  - Voller C-Sprachumfang

Zusammenfassung

- Why: Multi-Language Multi-Prover Verification Condition
- Grundlage: Typtheorie, Effekte
- Erzeugt Beweise
- Generiert Beweisverpflichtungen für Coq, PVS, Isabelle etc.
- Intern: Imperativ-funktionale Sprache WL
- Frontends für C, Java — Modellierung des Speichers
- Webseite: `http://why.lri.fr`

Vorlesung vom 04.07.06: Programmkonstruktion I

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare
- Korrekte Programmentwicklung
  - Programmkonstruktion
  - Programmentwicklung durch Transformation, Algorithmendesign

Programmkonstruktion

- Idee: Programm immer zusammen mit Korrektheitsbeweis entwicklen
- Genau Spezifikation nötig
- Formalisierung des Problems, Vor- und Nachbedingung
- Problem: Gegeben A, B. Finde P, so dass

$$\models \{A\}P\{B\}$$

Programmkonstruktion

- Idee: Programm immer zusammen mit Korrektheitsbeweis entwicklen
- Genau Spezifikation nötig
- Formalisierung des Problems, Vor- und Nachbedingung
- Problem: Gegeben A, B. Finde P, so dass

$$\models \{A\}P\{B\}$$

- Lösung: Rückwärtsbeweis mit Hoare-Regeln

Drei einfache Prinzipien

1. Sequentielle Zerlegung

Drei einfache Prinzipien

1. Sequentielle Zerlegung

2. Fallunterscheidung

Drei einfache Prinzipien

1. Sequentielle Zerlegung
2. Fallunterscheidung
3. Abschwächung der Vorbedingung, Stärkung der Nachbedingung

Drei einfache Prinzipien

1. Sequentielle Zerlegung
2. Fallunterscheidung
3. Abschwächung der Vorbedingung, Stärkung der Nachbedingung

- Problem: Schleifen und Invarianten

Schleifen und Invarianten

- Gesucht: P so dass $P \models \{A\}P\{B\}$

1. Finde I, c so dass $I \wedge \neg c \Longrightarrow B$
2. Zeige $A \Longrightarrow I$
3. Finde b so dass $\{I \wedge c\}b\{I\}$

- Dann: $\models \{A\}\texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}\{B\}$

Vier Wege, ein Prädikat abzuschwächen

1. Konjunkt streichen:

$$A \wedge B \wedge C \Longrightarrow A \wedge B$$

2. Konstante durch Variable ersetzen:

$$x < b[1:10] \quad \Longrightarrow \quad x < b[1:i] \wedge 1 \leq i \leq 10$$

3. Wertebereich vergrößern:

$$5 \leq i < 10 \Longrightarrow 0 \leq i < 10$$

4. Disjunkt hinzufügen:

$$A \Longrightarrow A \vee B$$

Beispiel I: Suche

- Generelles Problem: finde i so dass $R_0(i) \Longrightarrow P(i)$
  - Notation: $E(i : R_0 : P)$

1. Abschwächung von R_0 zu R:

$$E(i : R : P) \Longleftrightarrow E(i : R_0 : P), A(i : R_0 : P)$$

   Termination mit $\neg R(i) \vee P(i)$

2. Test vereinfachen: suche Q, R_0', P' so dass

$$E(i : R_0 : P) = Q(V(i : R_0' : P'))$$

- Beispiel: lineare Suche, binäre Suche

Beispiel II: Sortieren

- Sortieren: $sort(a, a_0) = perm(a, a_0) \wedge ordered(a_0)$
- Sei $occ(a, x) = card(\{j \mid 0 \leq j < n \Longrightarrow a[j] = x\})$, dann

$$perm(a, a_0) = \forall i.0 \leq i < n \Longrightarrow \begin{array}{l} occ(a, a[i]) = occ(a_0, a[i]) \wedge \\ occ(a, a_0[i]) = occ(a_0, a_0[i]) \end{array}$$

- Permutation is transitiv und symmetrisch
- Invariante: Vertauschen erhält Permutation
- Einfaches Beispiel: Dijkstra's Dutch National Flag

Zusammenfassung

- Programmkonstruktion: Programm zusammen mit Korrektheitsbeweis entwickeln
- Rückwärtsbeweis mit den Regeln des Hoare-Kalküls
- Einfach: Sequenzierung, Fallunterscheidung, Zuweisung
- Schwierig: Schleifen
  - Zerlegung in Invariante und Schleifenbedingung
- Beispiele: arithemtische Probleme, Suche, Sortieren
- Nächste Woche: fortgeschrittene Techniken — mehr Sortieren

Vorlesung vom 11.07.06: Programmkonstruktion II

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare
- Programmentwicklung
  - Programmkonstruktion
  - Programmentwicklung durch Transformation, Algorithmendesign

Programmentwicklung durch Transformation

- Von der Spezifikation zum Code:

$$SP_1 \rightsquigarrow SP_2 \rightsquigarrow \ldots \rightsquigarrow SP_n$$

- Einzelne Schritte:
- Modellverfeinerung (letztes Semester)

Programmentwicklung durch Transformation

- Von der Spezifikation zum Code:

$$SP_1 \rightsquigarrow SP_2 \rightsquigarrow \ldots \rightsquigarrow SP_n$$

- Einzelne Schritte:
- Modellverfeinerung (letztes Semester)
- Funktionen implementieren:

  Rekursion $\longrightarrow$ Endrekursion $\longrightarrow$ Iteration (letztes Semester)

Programmentwicklung durch Transformation

- Von der Spezifikation zum Code:

$$SP_1 \rightsquigarrow SP_2 \rightsquigarrow \ldots \rightsquigarrow SP_n$$

- Einzelne Schritte:
- Modellverfeinerung (letztes Semester)
- Funktionen implementieren:

  Rekursion $\longrightarrow$ Endrekursion $\longrightarrow$ Iteration (letztes Semester)
- Programmentwurfstransformationen
  - Einführung von Rekursion (heute)

Programmentwurfstransformationen

- Idee: Strukturierung der Spezifikation
  - Zerlegung
- Beispieltransformationen:
  - Divide & Conquer
  - Split of Postcondition
  - Global Search

Divide and Conquer

- Ziel: Funktion $f :: D\_F \Rightarrow R\_F$ implementieren
- Typen: D_F, R_F, D_G, and R_G
- Spezifikation als Vor- und Nachbedingung

```
F_pre    :: "D_F ⇒ bool"
F_post   :: "D_F ⇒ R_F ⇒ bool"
```

- Formalisierung als parametrisierte Spezifikation
  - Parameter: Typen, Operationen, Axiome

Divide and Conquer

- Operationen:

```
Measure      :: "D_F ⇒ W"
Primitive    :: "D_F ⇒ bool"
Dir_Solve    :: "D_F ⇒ R_F"
Decompose    :: "D_F ⇒ (D_G × D_F × D_F)"
Compose      :: "(R_G × R_F × R_F) ⇒ R_F"
O_COMPOSE    :: "(R_F × R_G × R_F × R_F) ⇒ bool"
O_DECOMPOSE  :: "(D_F × D_G × D_F × D_F) ⇒ bool"
G            :: "D_G ⇒ R_G"
G_pre        :: "D_G ⇒ bool"
G_post       :: "D_G ⇒ R_G ⇒ bool"
W_r         :: "(W × W) set"
```

Generische Implementation

- Generische Implementation von *F*:

```
consts
  F :: "D_F \<Rightarrow> R_F"
recdef F "inv_image W_r Measure"
 "F x = (if (Primitive x) then (Dir_Solve x)
          else (Compose o (f_product G F F) o Decompose) x)"
        (hints recdef_wf: A_WF)
```

- Generische Korrektheit von *F*:
  theorem *DaC* : **assumes** *"F_pre x"* **shows** *"F_post x (F x)"*
  - Voraussetzungen als Axiome
  - Beweis durch Instantiierung

Anwendung der Transformationsregel

- Anwendungsgebiet: Sortieralgorithmen
- Quicksort

Anwendung der Transformationsregel

- Anwendungsgebiet: Sortieralgorithmen
- Quicksort
- Mergesort

Anwendung der Transformationsregel

- Anwendungsgebiet: Sortieralgorithmen
- Quicksort
- Mergesort
- Insertsort
  - Insert-Funktion weitere Anwendung

Zusammenfassung

- Programmentwicklung durch Transformationen
- Programmentwurfstransformationen formalisieren Entwicklungsentscheidungen
- Beispiel: Divide-&-Conquer für
  - Quicksort, Merge, Insertsort
- Aber: hochspezielles Anwendungsgebiet

Vorlesung vom 18.07.06: Rückblick, Überblick, Ausblick

Wo sind wir?

- Am Ende . . .

Wo sind wir?

- Am Ende . . .
- Heute: Blick in Industrie und Forschung
  - Hardwareverifikation
  - Softwareverifikation
- Ausblick: Forschungsaktivitäten hier

Hardwareverifikation

- Intel IA-32 Pentium®4:
  - Register Renaming
  - BUS recycle logic
  - Floating point division and square root

Die HW-Verifikationsumgebung Forte

- `FL` (stark getypt, funktional, ML): Eingabe- und Skriptsprache
- Binary decision diagrams (BDDs) als first-class objects
- Symbolic trajectory evaluation als integrierte Funktionen
  - Aussagen der Form $\models_{ckt} [ant \Rightarrow cons]$
  - Konjunktion von Basisaussagen der Form $\mathtt{N}^t(n\ \mathtt{is}\ v\ \mathtt{when}\ g)$
  - Simulation in Verband $Bool + \{\mathtt{X}, \top\}$
- Theorembeweiser `ThmTac`
  - Klassische Logik höherer Ordnung, LCF-design
  - Aussagen der Form $\{\phi_{in}\}(tr_{in}, ckt, tr_{out}\{\phi_{out}\}$
- 15000 Zeilen für eine Mikroinstruktion, 45000 Zeilen für alle 20

HW-Verifikation — Status

- FDIV-Bug (1994), Kosten ca. US$ 500 Mill
- Verifikation amortisiert sich
- Auch andere Hersteller (AMD, IBM, HP/Compaq, . . .)
- Große Stückzahlen, spätere Änderung sehr teuer
- Korrektheit wird erwartet
- Verifikation Stand der Kunst

Software-Verifikation

- Software-Modelchecking
- Entscheidungsprozeduren
- Statische und dynamische Analyse
- Interaktive Verifikation

Software-Modelchecking

- Model-Checking (SMV, SPIN, FDR, . . .)
- Kombiniert mit Abstraktion (Datenabstraktion, Prädikatenabstraktion: Bandera, Blast,
- Vollautomatisch, findet (nur) Gegenbeispiele
- Geeignet für: nebenläufige Systeme, eingeschränker Zustandsraum, Protokolverifikation,

Automatische Werkzeuge

- Entscheidungsprozeduren: Chaff/zchaff, TSAT, Simplify, UCLID, SVC
  - Entscheidbarkeit von logischen Ausdrücken (SAT)
  - Arithmetik, Felder, abstrakte Datentypen
  - Vollautomatisch aber beschränktes Einsatzgebiet, Kombination, keine Gegenbeispiele, Korrektheit?
- Statische und Dynamische Analyse: BANE, Ccured; Purify
  - Gegen Pufferüberläufe, Speicherlecks, usw.
  - Skaliert gut — Korrektheit?

Verifikation

- Interaktives Beweisen: Hoare-Logik oder JML-artige
- Spezielle Verifikationswerkzeuge
  - Älter: KIV, VSE
  - Neuer: WHY, Spec#
- Universelle Theorembeweiser
  - STep, PVS, ACL2, Maude, HOL, Coq, Nuprl, Isabelle
- Aufwändig, aber korrekt

SW-Verifikation — Status

- Stand der Kunst: Verifikation für kritische Einzelfälle, Fallstudien
- Kein universelles Verifikationswerkzeug
  - Unterschiedliche Domänentheorie, unterschiedliche Spezifikationssprachen
- SW-Verifikation: Forschung — HW-Verifikation: Produktion

SW-Verifikation — Status

- Stand der Kunst: Verifikation für kritische Einzelfälle, Fallstudien
- Kein universelles Verifikationswerkzeug
  - Unterschiedliche Domänentheorie, unterschiedliche Spezifikationssprachen
- SW-Verifikation: Forschung — HW-Verifikation: Produktion
- Sofware kann leicht nachgebessert werden
- Software-Fehler werden akzeptiert
- Neue Einsatzgebiete werden Stand der Kunst vorantreiben
  - Eingebette und hybdride Systeme

Das SAMS-Projekt

- Ausgangspunkt: Der autonome Bremer Rollstuhl Rolland
- Idee: Sicherheitsmodul parametrisierbar für Roboter
- Sicherheitsmodul integriert in Laserscanner, für autonome Roboter, fahrerlose Transportsysteme
- Ziel: TÜV-Zulassung, formale Verifikation des Sicherheitsmoduls
- DFKI Bremen, Uni Bremen, Leuze lumiflex — 3 Jahre

Das AWE-Projekt

- Ausgangspunkt: Wiederverwendung des gesamten Entwicklungsprozesses, nicht nur der Artefakte
- Idee: Formale Entwicklungen als formale Objekte — systematische Verallgemeinerung
- abstraction for reuse
- Implementierung basierend auf Isabelle

Zusammenfassung

- Grundlagen der Programmverifikation
  - Vollständigkeit des Hoare-Kalküls
  - Generierung von Verifikationsbedingungen
- Jenseits des Hoare-Kalküls
  - Java Modelling Language (ähnlich: OCL)
  - Werkzeugunterstützung: WHY
- Programmkonstruktion:
  - Programmentwicklung als Rückwärtsbeweis
  - Programmentwicklung durch Transformation (divide-&-conquer)

Und jetzt?

- Mitarbeit in AG BKB/DFKI Lab Bremen
  - Studentische Hilfskräfte
  - Diplomarbeiten
- Tutor PI3, anybody?