Kapitel III

Programmanalyse und Typsysteme

Motivation

Verifikationsmethoden wie Model Checking untersuchen den vollständigen Zustandsraum eines Systems.

- Das beschränkt die Verifikation auf (Teil-)Systeme mit nicht zu großem Zustandsraum.
- Der Zustandsraum kann durch Abstraktion verkleinert werden; dann muss aber auch die Abstraktion verifiziert werden.

In diesem Kapitel befassen wir uns mit Methoden der Programmiersprachentheorie zur Analyse von Software realistischer Größe.

Motivation

Statische Analyse von Programmen realistischer Größe

Schwächere Eigenschaften können automatisch überprüft werden.

- Abwesenheit von Laufzeitfehlern wie Division durch Null, falscher Speicherzugriff, ungefangene Exceptions, Verletzung von Ressourcenschranken und -protokollen (z.B. Dateizugriff).
- Datenflusseigenschaften zur Programmoptimierung

Solche Eigenschaften sind z.B. im Compilerbau nützlich.

Kompliziertere Eigenschaften erfordern manuelle Hilfe durch

- Angabe von Typannotaten,
- Spezifikation von Zusicherungen und Invarianten,
- formale Beweise.

Inhalt Kapitel III

- Induktive Definitionen
- Programmanalyse für imperative Programme
  - Spezifikation einer While-Sprache
  - Datenflussanalyse
- Fixpunkttheorie
- Programmanalyse und Typsysteme
- Typ- und Effektsysteme
  - Funktionale Sprache
  - Typinferenz
  - Polymorphie
  - Kontrollflussanalyse

Induktive Definitionen

In der Programmiersprachentheorie werden viele Konzepte durch induktive Definitionen formalisiert.

Das übliche Format für induktive Definitionen sind *Inferenzregeln*.

Beispiele aus der Vorlesung:

$$\frac{e_1 \longrightarrow \mathsf{fn}_\pi\ x \Rightarrow e_1' \qquad e_2 \longrightarrow v_2 \qquad e_1'[x \mapsto v_2] \longrightarrow v}{e_1\ e_2 \longrightarrow v}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \xrightarrow{\varphi_1} \tau_2\ \&\ \varphi_2 \qquad \Gamma \vdash e_2 : \tau_1\ \&\ \varphi_3}{\Gamma \vdash e_1\ e_2 : \tau_2\ \&\ \varphi_1 \cup \varphi_2 \cup \varphi_3}$$

Urteile

Ein *Urteil* ist ein formaler Ausdruck, der eine bestimme Aussage ausdrückt (meist eine Eigenschaft bestimmter Objekte).

Beispiele:

- n *even* — "Die natürliche Zahl n ist gerade."
- n *odd* — "Die natürliche Zahl n ist ungerade."
- $\mathit{sum}(m, n) = r$ — "Die Summe der nat. Zahlen m und n ist die nat. Zahl r."
- $\phi \vdash \psi$ — "Jede Belegung, welche die aussagenlogischen Formeln ϕ wahr macht, macht auch ψ wahr."
- $e : X$ — "Der Programmausdruck e hat Typ X."

Urteile sind als *rein syntaktische* Ausdrücke ohne inhärente Bedeutung zu verstehen (oben sind "*sum*" und "$\vdash$" nur Symbole).

Inferenzregeln

Die Bedeutung von Urteilen wird durch *Inferenzregeln* festgelegt.

Eine Inferenzregel hat die Form:

$$(\textsc{name})\ \frac{U_1 \qquad U_2 \qquad \ldots \qquad U_n}{U}$$

- Die Urteile $U_1, \ldots, U_n$ heißen *Prämissen*.
- Das Urteil U heißt *Konklusion*.
- Inferenzregeln ohne Prämissen (d.h. mit $n = 0$) heißen *Axiome*.

Bedeutung der Regel: Wenn die Prämissen alle zutreffen, dann auch die Konklusion.

Inferenzregeln: Beispiele

Gerade/ungerade:

$$\text{(EvenZ)}\ \frac{}{0\ \mathit{even}} \qquad \text{(OddS)}\ \frac{n\ \mathit{even}}{n+1\ \mathit{odd}} \qquad \text{(EvenS)}\ \frac{n\ \mathit{odd}}{n+1\ \mathit{even}}$$

Summen:

$$\text{(SumZ)}\ \frac{}{\mathit{sum}(n,0) = n} \qquad \text{(SumS)}\ \frac{\mathit{sum}(n,m) = r}{\mathit{sum}(n,m+1) = r+1}$$

Aussagenlogik:

$$\frac{}{\phi \vdash \phi} \qquad \frac{\phi \vdash \psi_1 \qquad \phi \vdash \psi_2}{\phi \vdash \psi_1 \wedge \psi_2} \qquad \frac{\phi \vdash \psi_1}{\phi \vdash \psi_1 \vee \psi_2} \qquad \frac{\phi \vdash \psi_2}{\phi \vdash \psi_1 \vee \psi_2}$$

$$\ldots$$

Inferenzregeln (genau)

Inferenzregeln enthalten meist Metavariablen und sind eigentlich als Regelschemata zu verstehen.

Gemeint sind alle Regeln, die man durch Einsetzung beliebiger passender(!) Ausdrücke für die Metavariablen erhält.

Beispiel:

$$\frac{\mathit{sum}(n, m) = r}{\mathit{sum}(n, m + 1) = r + 1}$$

steht für

$$\frac{\mathit{sum}(0, 0) = 0}{\mathit{sum}(0, 1) = 1}, \frac{\mathit{sum}(0, 0) = 1}{\mathit{sum}(0, 1) = 2}, \ldots, \frac{\mathit{sum}(5, 6) = 18}{\mathit{sum}(5, 7) = 19}, \ldots$$

Für m, n und r dürfen nur nat. Zahlen eingesetzt werden, da die Urteile in der Regel nur für nat. Zahlen definiert sind.

Inferenzregeln (genau)

Beispiel:

$$\frac{\phi \vdash \psi_1 \qquad \phi \vdash \psi_2}{\phi \vdash \psi_1 \wedge \psi_2}$$

steht für alle Regeln, die man durch Einsetzung von konkreten aussagenlogischen Formeln für ϕ, ψ_1 und ψ_2 erhält, z.B.

$$\frac{\top \vdash \top \qquad \top \vdash \top}{\top \vdash \top \wedge \top}, \frac{(A \Rightarrow B) \vdash (C \wedge B) \qquad (A \Rightarrow B) \vdash \bot}{(A \Rightarrow B) \vdash (C \wedge B) \wedge \bot}, \ldots$$

Hier dürfen nur aussagenlogische Formeln eingesetzt werden, da die Urteile nur dafür definiert sind.

Inferenzregeln: Seitenbedingungen

Die möglichen Werte der Metavariablen in schematischen Regeln werden manchmal durch Seitenbedingungen eingeschränkt.

Beispiel: Die Regel (SumS) könnte auch so geschrieben werden.

$$(\textsc{SumS})\ \frac{\mathit{sum}(n, m) = r}{\mathit{sum}(n, m') = r'}\ m' = m + 1,\ r' = r + 1$$

Die Metavariablen dürfen nur so instantiiert werden, dass alle Seitenbedingungen erfüllt sind.

(Seitenbedingungen werden manchmal auch wie Prämissen über den Strich geschrieben.)

Herleitungen

Der Begriff der *Herleitung für ein Urteil U* ist induktiv wie folgt definiert.

- Jede Regel der Form (d.h. ohne Prämisse)

  $$(\mathsf{R})\ \frac{}{U}$$

  ist eine Herleitung für ihre Konklusion U.
- Gibt es eine Regel

  $$(\mathsf{S})\ \frac{U_1 \qquad U_2 \qquad \ldots \qquad U_n}{U}$$

  und sind $\Pi_1, \ldots, \Pi_n$ jeweils Herleitungen für $U_1, \ldots, U_n$, dann ist

  $$(\mathsf{S})\ \frac{\Pi_1 \qquad \Pi_2 \qquad \ldots \qquad \Pi_n}{U}$$

  eine Herleitung für U.
- Nichts weiter ist eine Herleitung.

Herleitungen: Beispiele

Beispiele:

$$\dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{\dfrac{}{A \vdash A} \qquad \dfrac{}{A \vdash B \vee A}}{A \vdash A \wedge (B \vee A)}}{A \vdash A \wedge (A \wedge (B \vee A))}$$

$$\begin{array}{l}(\textsc{SumZ})\ \dfrac{}{sum(3,0) = 3} \\ (\textsc{SumS})\ \dfrac{sum(3,0) = 3}{sum(3,1) = 4} \\ (\textsc{SumS})\ \dfrac{sum(3,1) = 4}{sum(3,2) = 5}\end{array}$$

Kein Beispiel:

$$\begin{array}{l}(\textsc{SumS})\ \dfrac{sum(3,0) = 4}{sum(3,1) = 5} \\ (\textsc{SumS})\ \dfrac{sum(3,1) = 5}{sum(3,2) = 6}\end{array}$$

(Regelnamen und Seitenbedingungen werden oft nicht ausgeschrieben.)

Induktive Definition

Inferenzregeln sind induktive Definitionen von Urteilen.

Induktionsprinzip: Um zu zeigen, dass alle herleitbaren Urteile eine bestimmte Eigenschaft haben, genügt es für alle Regeln zu zeigen: Wenn die Prämissen der Regel die Eigenschaft haben, dann auch die Konklusion.

Beispiel: Wenn $\mathsf{sum}(m, n) = r$ herleitbar ist, dann gilt $r = m + n$.

- Regel (SumZ)
$$\frac{}{\mathit{sum}(n, 0) = n}$$
Es gilt $n = n + 0$, also hat die Konklusion die Eigenschaft.
- Regel (SumS)
$$\frac{\mathit{sum}(n, m) = r}{\mathit{sum}(n, m + 1) = r + 1}$$
Angenommen die Prämisse hat die Eigenschaft, d.h. $r = m + n$. Dann gilt $r + 1 = m + n + 1 = (m + 1) + n$, also hat auch die Konklusion die Eigenschaft.

Induktive Definitionen

Inferenzregeln sind als reine Spezifikation von Urteilen zu verstehen.

- Inferenzregeln spezifizieren was herleitbar ist, aber nicht unbedingt wie.
- Die Frage, ob ein Urteil herleitbar ist, kann je nach Art der Regeln sehr leicht oder sehr schwer sein.
- Man verwendet Inferenzregeln, um möglichst einfach zu spezifizieren, was eine Analyse leisten soll, ohne sich dabei bereits auf einen Analysealgorithmus festzulegen.
- Beispiel: Das Urteil $\mathit{sum}(n, m) = r$ sagt, was eine Summe ist; verschiedene Implementierungen der Addition realisieren diese Spezifikation, z.B.
  - Ripple-Carry Adder
  - Lookahead-Carry-Adder

Einfache While-Sprache

Arithmetische Ausdrücke:

$$a, a_1, a_2 ::= x \mid n \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1/a_2$$

Boolesche Ausdrücke:

$$b, b_1, b_2 ::= \texttt{true} \mid \texttt{false} \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b \mid a_1 < a_2 \mid a_1 = a_2$$

Programmstücke (statements):

$$S, S_1, S_2 ::= [x := a]^\ell \mid [\texttt{skip}]^\ell \mid S_1; S_2 \mid \texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } [b]^\ell \texttt{ do } S$$

Hierbei x läuft über Programmvariablen, n über Integerkonstanten, ℓ über Labels (konkret: natürliche Zahlen), welche elementare Programmteile eindeutig markieren sollen.
Konvention: $S_1; S_2; S_3$ steht für $S_1; (S_2; S_3)$.

Beispielprogramm

$$[\texttt{y:=x}]^1; [\texttt{z:=1}]^2; \texttt{while } [\texttt{1<y}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6$$

Alternative Notation:

```
1: y:=x;
2: z:=1;
3: while 1<y do (
4:     z:=z*y;
5:     y:=y-1);
6: y:=0
```

Operationelle Semantik

Die operationelle Semantik spezifiziert wie sich While-Programme bei der Ausführung verhalten.

Ein *Programmzustand* (σ) ist eine endliche Abbildung von Programmvariablen auf ganze Zahlen.

Beispiel: $\sigma = [x \mapsto 3, y \mapsto 4]$.

Wenn σ alle Variablen in einem arithmetischen Ausdruck a auf Werte abbildet, dann schreiben wir $[\![a]\!]\sigma$ für den Wert des Ausdrucks mit den entsprechenden Belegungen. Sonst ist $[\![a]\!]\sigma$ undefiniert.

Beispiel: $[\![x + 4 * y]\!]\sigma = 19$

Analog für Boolesche Ausdrücke.

Beispiel: $[\![x + 4 * y < 18]\!]\sigma = \mathit{false}$

Operationelle Semantik

Die operationelle Semantik ist durch zwei Urteile gegeben.

- $\langle S, \sigma \rangle \to \sigma'$ "Die Ausführung von S im Startzustand σ ist nach einem Schritt beendet und endet im Zustand σ'."
- $\langle S, \sigma \rangle \to \langle S', \sigma' \rangle$ "Die Ausführung von S im Anfangszustand σ führt nach einem Schritt zum Zwischenzustand σ'; es bleibt danach noch die Anweisung S' abzuarbeiten."

Regeln für die operationelle Semantik

$$(\textsc{ass})\ \frac{}{\langle [x:=a]^\ell, \sigma\rangle \to \sigma[x \mapsto [\![a]\!]\sigma]}$$

$$(\textsc{skip})\ \frac{}{\langle \mathtt{skip}, \sigma\rangle \to \sigma}$$

$$(\textsc{seq1})\ \frac{\langle S_1, \sigma\rangle \to \langle S_1', \sigma'\rangle}{\langle S_1; S_2, \sigma\rangle \to \langle S_1'; S_2, \sigma'\rangle}$$

$$(\textsc{seq2})\ \frac{\langle S_1, \sigma\rangle \to \sigma'}{\langle S_1; S_2, \sigma\rangle \to \langle S_2, \sigma'\rangle}$$

Beachte: (ASS) ist nur anwendbar, wenn $[\![a]\!]\sigma$ definiert ist.

Regeln für die operationelle Semantik, Forts.

$$(\textsc{IfT})\ \frac{}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, \sigma\rangle \to \langle S_1, \sigma\rangle}\ [\![b]\!]\sigma = \mathit{true}$$

$$(\textsc{IfF})\ \frac{}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, \sigma\rangle \to \langle S_2, \sigma\rangle}\ [\![b]\!]\sigma = \mathit{false}$$

$$(\textsc{WhileT})\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ S, \sigma\rangle \to \langle S; \texttt{while}\ b\ \texttt{do}\ S, \sigma\rangle}\ [\![b]\!]\sigma = \mathit{true}$$

$$(\textsc{WhileF})\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ S, \sigma\rangle \to \sigma}\ [\![b]\!]\sigma = \mathit{false}$$

Beispiel

Schreibe S als Abkürzung für folgendes Programm.

$$\texttt{while}\ [x < 1]^1\ \text{do}\ ([y := y - 1]^2;\ [x := x - 1]^3)$$

Dann kann man folgende Urteile herleiten:

- $\langle S, \rho_1 \rangle \to \langle ([y := y - 1]^2;\ [x := x - 1]^3); S, \rho_1 \rangle$
- $\langle ([y := y - 1]^2;\ [x := x - 1]^3); S, \rho_1 \rangle \to \langle [x := x - 1]^3; S, \rho_2 \rangle$
- $\langle [x := x - 1]^3; S, \rho_2 \rangle \to \langle S, \rho_3 \rangle$
- $\langle S, \rho_3 \rangle \to \rho_3$

wobei $\rho_1 := [x \mapsto 1, y \mapsto 5]$ und $\rho_2 := [x \mapsto 1, y \mapsto 4]$ und $\rho_3 := [x \mapsto 0, y \mapsto 4]$.

Herleitung für das zweite Urteil:

$$\text{(SEQ1)}\ \frac{\text{(SEQ2)}\ \dfrac{\text{(ASS)}\ \dfrac{}{\langle [y := y - 1]^2, \rho_1 \rangle \to \rho_2}}{\langle [y := y - 1]^2;\ [x := x - 1]^3, \rho_1 \rangle \to \langle [x := x - 1]^3, \rho_2 \rangle}}{\langle ([y := y - 1]^2;\ [x := x - 1]^3); S, \rho_1 \rangle \to \langle [x := x - 1]^3; S, \rho_2 \rangle}$$

Operationelle Semantik

Die operationelle Semantik ist eine kompakte Spezifikation der Programmausführung.

- Die Inferenzregeln selbst spezifizieren noch keinen Algorithmus zur Programmausführung.
- Ein Compiler/Interpreter implementiert sollte die operationelle Semantik möglichst effizient implementieren.
- Die operationelle Semantik ist Spezifikation des Compilers/Interpreters.
- Man kann einen einfachen Interpreter für die Sprache von der operationellen Semantik ableiten.
  Für gegebene S und σ versucht man einen Herleitunsbaum für $\langle S, \sigma \rangle \to X$ von unten nach oben aufzubauen und damit X zu bestimmen.

Small-Step- und Big-Step-Semantik

Die Urteile $\langle S, \sigma \rangle \to \sigma'$ und $\langle S, \sigma \rangle \to \langle S', \sigma' \rangle$ machen Aussagen über einen einzigen Schritt der Programmausführung.
Das nennt man eine *Small-Step-Semantik*.

Interessiert man sich nur für die Ausführung des gesamten Programms, kann man auch ein Urteil für eine *Big-Step-Semantik* definieren:

$$\text{(BIG1)}\ \frac{\langle S, \sigma \rangle \to \sigma'}{\langle S, \sigma \rangle \Downarrow \sigma'}$$

$$\text{(BIG2)}\ \frac{\langle S, \sigma \rangle \to \langle S', \sigma' \rangle \qquad \langle S', \sigma' \rangle \Downarrow \sigma''}{\langle S, \sigma \rangle \Downarrow \sigma''}$$

Das Urteil $\langle S, \sigma \rangle \Downarrow \sigma'$ sagt: Wenn man S mit Anfangszustand σ laufen lässt, dann terminiert die Programmausführung im Endzustand σ'.