Programmiersprachen

Vorlesung 3 vom 23.10.23

Einfache Typen, Anweisungen und Seiteneffekte

Christoph Lüth

Universität Bremen

Wintersemester 2023/24

Organisatorisches

- Übungen: 8 Repos, Dateiaustauschtest
- Bitte `uebung-XX.md` nicht umbenennen.
- Vorlesung am 30.10.2023 **findet statt**.

Wo sind wir?

- Einführung
- Einfache Ausdrücke und Werte
- Einfache Typen, Anweisungen und Seiteneffekte
- Variablen und Speichermodelle
- Aggregierende Typen
- Ausnahmen und Fehlerbehandlung
- Prozeduren und Funktionen
- Fortgeschrittene Typsysteme
- Datenabstraktion
- Programmierparadigmen
- Skriptsprachen
- Ab Woche 11 (2024): Studentische Vorträge.

Typen

Warum Typen?

- Evaluation kann “stecken bleiben” (Bsp. $x/0$).

Übung 2.1: Aus welchen Gründen reduziert ein Ausdruck nicht zu einem Wert?

Warum Typen?

- Evaluation kann “stecken bleiben” (Bsp. $x/0$).

Übung 2.1: Aus welchen Gründen reduziert ein Ausdruck nicht zu einem Wert?

- Wie können wir das verhindern?

- Durch **Typisierung**

Typüberprüfung

- **Statische** Typsysteme:
  - Typen werden zur Compile-Zeit geprüft.
  - Zur Laufzeit werden keine Typinformationen benötigt (“type erasure”)
  - Sicher und effizient, aber unflexibel
  - Bsp: C, Rust, Haskell, Java
- **Dynamische** Typsysteme
  - Typen werden zur Laufzeit geprüft.
  - Flexibel, aber fehleranfällig.
  - Bsp: Python, Java (dynamische Bindung von Methoden)

Implizit vs. explizit

- **Explizit** oder manifeste Typsysteme: alle Typen werden **explizit** angegeben
  - Compiler muss nur prüfen
  - Beispiel: Java
- **Implizite** Typsysteme: Typen werden abgeleitet
  - Compiler muss Typ inferieren
  - Beispiel: Rust, Haskell, Hindley-Milner-Typsystem
  - Problem: Entscheidbarkeit, bspw. mit Subtyping unentscheidbar

Frage: Ist Python implizit oder explizit?

Typinferenz

- *type judgements* sind von der Form $\Gamma \vdash e : t$
- $\Gamma \in \mathbf{Idt} \rightharpoonup \mathcal{T}ypes$ ist eine **Typumgebung**
- e ist ein Ausdruck
- t ist ein Typ $t \in \mathcal{T}ypes = \{\mathbf{int}, \mathbf{bool}, \mathbf{unit}\}$
- *type judgements* werden durch **Typinferenz** hergeleitet

Regeln für die Typinferenz

$$\frac{}{\Gamma \vdash n : \textbf{int}} \qquad \frac{x \in \mathrm{dom}(\Gamma), \Gamma(x) = T}{\Gamma \vdash x : T}$$

$$\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 + e_2 : \textbf{int}} \qquad \frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 - e_2 : \textbf{int}}$$

$$\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 * e_2 : \textbf{int}} \qquad \frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 / e_2 : \textbf{int}}$$

$$\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 == e_2 : \textbf{bool}} \qquad \frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 < e_2 : \textbf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \textbf{bool} \qquad \Gamma \vdash e_2 : \textbf{bool}}{\Gamma \vdash e_1 \,\&\&\, e_2 : \textbf{bool}} \qquad \frac{\Gamma \vdash e_1 : \textbf{bool} \qquad \Gamma \vdash e_2 : \textbf{bool}}{\Gamma \vdash e_1 \,||\, e_2 : \textbf{bool}} \qquad \frac{\Gamma \vdash e : \textbf{bool}}{\Gamma \vdash !\, e : \textbf{bool}}$$

Eigenschaften der Typinferenz

- Typinferenz ist **entscheidbar**.

- Was bedeuten folgende Eigenschaften (und gelten sie)?

$$\forall \Gamma, \sigma, e, t. \Gamma \vdash e : t \Longrightarrow \exists v. \langle e, \sigma \rangle \to v \tag{1}$$

$$\Gamma \vdash e : \textbf{int} \wedge \langle e, \sigma \rangle \to v \Longrightarrow v \in \mathbb{Z} \tag{2}$$

Anweisungen

Einfache Anweisungen:

- Kernsprache:
  - Zuweisung
  - Sequenzierung und leere Anweisung — **Sequenzen** von Anweisungen
  - Fallunterscheidung
  - Iteration
    - `while`, `repeat`, Rekursion
  - $\Longrightarrow$ Turing-mächtig

Mehr Anweisungen

- `for`-Schleifen:
  - C, Java: syntaktischer Zucker für `while`
  - Rust, Python: **Iterator**

```
m = {'a' : 1, 'b' : 5, 'c': 7 }
for k in m:
  print(k)
```

```
let array = [1u32, 3, 3, 7];
for i in array.iter() {
    println!("> {}", i);
}
```

Mehr Anweisungen

- Sprünge: `goto` etc. — "considered harmful"[1]
  - C kennt `goto` und `longjmp()`
  - Java und Rust haben Labels
- Manche Sprachen unterscheiden Ausdrücke und Anweisungen nicht
  - In C sind Zuweisungen Ausdrücke, und Ausdrücke Anweisungen
  - In Haskell ist alles ein Ausdruck

[1] Edsger W. Dijkstra. 1968. Letters to the editor: go to statement considered harmful. Commun. ACM 11, 3 (March 1968), 147-148. https://doi.org/10.1145/362929.362947

Operationale Semantik

Abstrakte Syntax

- Anweisungen:

$$
\begin{aligned}
c ::= \ & \mathbf{Idt} := \mathbf{Exp} \\
\mid \ & c_1; c_2 \\
\mid \ & \mathbf{nil} \\
\mid \ & \mathbf{if}\ (e)\ \mathbf{then}\ c_1\ \ \mathbf{else}\ \ c_2 \\
\mid \ & \mathbf{while}\ (e)\ c
\end{aligned}
$$

- Operationale Semantik: $\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

Regeln der operationalen Semantik I

- Regeln:

$$\frac{\langle e, \sigma\rangle \to_{Exp} n \qquad n \in \mathbb{Z}}{\langle x := e, \sigma\rangle \to_{Stmt} \sigma[x \mapsto n]} \qquad \frac{}{\langle \textbf{nil}, \sigma\rangle \to_{Stmt} \sigma}$$

$$\frac{\langle c_1, \sigma\rangle \to_{Stmt} \sigma' \qquad \langle c_2, \sigma'\rangle \to_{Stmt} \sigma''}{\langle c_1; c_2, \sigma\rangle \to_{Stmt} \sigma''}$$

- Beispiel: $\sigma \stackrel{def}{=} \langle\rangle$

```
x := 6;
y := 4+ x;
```

Regeln der operationalen Semantik II

- Regeln:

$$\frac{\langle b, \sigma\rangle \to_{Lexp} \mathit{true} \qquad \langle c_1, \sigma\rangle \to_{Stmt} \sigma'}{\langle \textbf{if } (b) \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \to_{Stmt} \sigma'} \qquad \frac{\langle b, \sigma\rangle \to_{Lexp} \mathit{false} \qquad \langle c_2, \sigma\rangle \to_{Stmt} \sigma'}{\langle \textbf{if } (b) \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \to_{Stmt} \sigma'}$$

- Beispiel: $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 10\rangle$

```
if (x != 0) {
  y := y/x;
  } else {
  y := 0;
  }
```

Regeln der operationalen Semantik III

- Regeln:

$$\frac{\langle b, \sigma\rangle \to_{Lexp} \mathit{false}}{\langle \textbf{while } (b)\ c, \sigma\rangle \to_{Stmt} \sigma}$$

$$\frac{\langle b, \sigma\rangle \to_{Lexp} \mathit{true} \qquad \langle c, \sigma\rangle \to_{Stmt} \sigma' \qquad \langle \textbf{while } (b)\ c, \sigma'\rangle \to_{Stmt} \sigma''}{\langle \textbf{while } (b)\ c, \sigma\rangle \to_{Stmt} \sigma''}$$

- Beispiel: $\sigma \stackrel{def}{=} \langle x \mapsto 3\rangle$

```
f := 1;
while (x > 0) {
  f := f* x;
  x := x-1;
}
```

Auswertung von Ausdrücken mit Seiteneffekten

- Imperative Sprachen haben Ausdrücke mit **Seiteneffekt**
- Auswertung verändert Zustand.
- Dazu **Änderung** der Sprache:

$$
\begin{aligned}
e ::= {} & \mathbb{Z} \mid \mathit{I} \mid \mathit{true} \mid \mathit{false} \\
& \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2 \\
& \mid e_1 == e_2 \mid e_1 < e_2 \\
& \mid \,!\, e \mid e_1 \,\&\&\, e_2 \mid e_1 \,||\, e_2 \\
& \mid \mathbf{Idt} := \mathbf{Exp}
\end{aligned}
$$

$$
c ::= e \mid \mathbf{if}\ (e)\ \mathbf{then}\ c_1\ \ \mathbf{else}\ \ c_2 \mid \mathbf{while}\ (e)\ c \mid c_1; c_2 \mid \mathbf{nil}
$$

Neue Regeln

Auswertung: $\langle e, \sigma \rangle \to_{Exp} \langle v, \sigma' \rangle$

$$\frac{i \in \mathbb{Z}}{\langle i, \sigma \rangle \to_{Exp} \langle i, \sigma \rangle} \qquad \frac{b \in \mathbb{B}}{\langle b, \sigma \rangle \to_{Exp} \langle b, \sigma \rangle} \qquad \frac{x \in \textbf{Idt}, x \in \mathrm{dom}(\sigma), \sigma(x) = v}{\langle x, \sigma \rangle \to_{Exp} \langle v, \sigma \rangle}$$

$$\frac{\langle e_1, \sigma \rangle \to_{Exp} \langle n_1, \sigma_1 \rangle \qquad \langle e_2, \sigma_1 \rangle \to_{Exp} \langle n_2, \sigma_2 \rangle \qquad n_i \in \mathbb{Z}}{\langle e_1 + e_2, \sigma \rangle \to_{Exp} \langle n_1 + n_2, \sigma_2 \rangle}$$

$$\frac{\langle e, \sigma \rangle \to_{Exp} \langle n, \sigma' \rangle \qquad n \in \mathbb{Z}}{\langle x := e, \sigma \rangle \to_{Exp} \langle n, \sigma'[x \mapsto n] \rangle}$$

$$\frac{\langle e, \sigma \rangle \to_{Exp} \langle v, \sigma' \rangle}{\langle e, \sigma \rangle \to_{Stmt} \sigma'}$$

Auswertung

- Was ist der **Wert** von $x := e$?
- In welcher Reihenfolge werden die Argumente von $+$ ausgewertet?
  - Gilt auch für $-$, $*$, $/$, aber **nicht** für &&, ||

Übung 2.6: Gegeben $\sigma = \langle x \mapsto 0, y \mapsto 0 \rangle$.
Berechne die Auswertung von `(x := y+1)+(y := x+5)`

Alternative Regeln

- Auswertung von rechts nach links:

$$\frac{\langle e_1, \sigma_1\rangle \to_{Exp} \langle n_1, \sigma_2\rangle \qquad \langle e_2, \sigma\rangle \to_{Exp} \langle n_2, \sigma_1\rangle \qquad n_i \in \mathbb{Z}}{\langle e_1 + e_2, \sigma\rangle \to_{Exp} \langle n_1 + n_2, \sigma_2\rangle}$$

- Unbestimmt:

$$\frac{\langle e_1, \sigma\rangle \to_{Exp} \langle n_1, \sigma_1\rangle \qquad \langle e_2, \sigma_1\rangle \to_{Exp} \langle n_2, \sigma_2\rangle \qquad n_i \in \mathbb{Z}}{\langle e_1 + e_2, \sigma\rangle \to_{Exp} \langle n_1 + n_2, \sigma_2\rangle}$$

$$\frac{\langle e_1, \sigma_1\rangle \to_{Exp} \langle n_1, \sigma_2\rangle \qquad \langle e_2, \sigma\rangle \to_{Exp} \langle n_2, \sigma_1\rangle \qquad n_i \in \mathbb{Z}}{\langle e_1 + e_2, \sigma\rangle \to_{Exp} \langle n_1 + n_2, \sigma_2\rangle}$$

Übung 2.7: In welcher Reihenfolge werden die Argumente mehrstelliger Operationen in C, Rust, Java, Python, Haskell ausgewertet?

Zusammenfassung

- Typsystem können implizit oder explizit, statisch oder dynamisch sein.
- Die Typableitung $\Gamma \vdash e : t$ wird durch **Regeln** definiert.
- Die Auswertung von **Anweisungen** erzeugt einen Zustandsübergang: $\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$
- Die Auswertung von Ausdrücken mit Seiteneffekten erzeugt einen Wert **und** einen neuen Zustand: $\langle e, \sigma \rangle \rightarrow_{Exp} \langle v, \sigma' \rangle$