Programmiersprachen
Vorlesung 6 vom 13.11.23
Ausnahmen und Fehlerbehandlung

Christoph Lüth

Universität Bremen

Wintersemester 2023/24

Organisatorisches

- Nächste Woche (20.11.): Stromabschaltung

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.

Ausnahmen (Exceptions)

- Motivation:
  - Fehlerbehandlung in geschachtelten Funktionen
  - Ohne Sprünge nur umständliche Fallunterscheidungen

Ausnahmen (Exceptions)

- Motivation:
  - Fehlerbehandlung in geschachtelten Funktionen
  - Ohne Sprünge nur umständliche Fallunterscheidungen
  - Nicht alle Fehlermöglichkeiten **können** im Vorfeld ausgeschlossen werden
  - Klassisches Beispiel: Dateizugriff

```
if os.path.exists(filenm):
  # someone deletes file
  fd= open(filenm) # FEHLER!
```

```
int main() {
  fd= open("data");
  ... P(fd); ...;
  close(fd);
  }

void P(int fd) { ... Q(fd); ... }

void Q(int fd) { ... R(fd); ... }

void R(int fd) {
  if ((x= read(fd, 1024))== -1)
    // Fehler!
  ...
  }
```

Ausnahmen

- Konzeptionell: Ausnahmen sind **Erweiterung des Definitionsbereichs**:

$$\texttt{f} : A \to B \texttt{ throws } C \;=\; \texttt{f} : A \to B + C$$

$$B[f(x)] \texttt{ catch } e \Rightarrow E \;=\; \begin{cases} B[b] & f(x) = b \\ E & f(x) = e \end{cases}$$

- Semantisch: **Programmfehler** sind "undeklarierte Ausnahmen"
  - Aber: nicht alle können **gefangen** werden
- Unterstützung von Ausnahmen durch eine Programmiersprache benötigt:
  - Ausnahmen **deklarieren** — meist eigener Typ (SML) oder Klasse (Java, Haskell)
  - Ausnahmen **auslösen** (`throw`, `raise`)
  - Ausnahmen **fangen** (`catch`)

Ausnahmen in Java

- Ausnahmen sind Objekte der Klassen `Throwable`, `Exception`
- Geworfene Ausnahmen müssen **deklariert** werden (`throws ...`)
  - Warum? Static Scoping (siehe Beispiel)
- Schema:

```
try
  block
catch (exception_type e)
  block
catch (exception_type e)
  block
finally
  block
```

Ausnahmen in Haskell

- Ausnahmen sind Instanzen der Typklasse `Exception` (Modul `Control.Exception`)
  - Situation in Haskell98 anders
- Werfen und fangen:

```
throw :: Exception e ⇒ e → a
catch :: Exception e ⇒ IO a → (e → IO a) → IO a
```

- Exceptions können überall geworfen werden, aber nur als Aktion (`IO`) gefangen werden.
  - Warum? Bricht referentielle Transparenz
- Durch Nicht-Striktheit (verzögerte Auswertung) werden Ausnahmen später geworfen als man denkt (siehe Beispiel)

Ausnahmen in C

- C hat **keine** Exceptions
- Alternativen:
  - `goto`
  - `setjmp` and `longjmp`
  - `switch`, siehe Duff's Device

Duff's Device:

```
void send(short *to, short *from,
          int count)
{
  int n = (count+ 7)/8;
  switch (count % 8) {
   case 0: do {
              *to = *from++;
   case 7:    *to = *from++;
   case 6:    *to = *from++;
   case 5:    *to = *from++;
   case 4:    *to = *from++;
   case 3:    *to = *from++;
   case 2:    *to = *from++;
   case 1:    *to = *from++;
            } while (--n > 0);
} }
```

Quelle: Wikipedia

Programmieren mit Ausnahmen

Ausnahmen sollten **unerwartete** und **seltene** Situationen modellieren.

1. "Ask forgiveness, not permission"
   - Bessere Robustheit
2. "Let it fail"
3. Nur Ausnahmen fangen, die auch behandelt werden
   - Unbehandelte Fehler werden nur schlimmer
4. Ausnahmen können der Effizienz und der Übersichtlichkeit dienen.

Semantik

Ausnahmen

- Einfacher Ansatz: es gibt eine feste Menge von Ausnahmen $X = \{x_1, \ldots, x_n\}$.
- Dazu Laufzeitfehler $E = \{E_0, E_1\}$ (Division durch Null, undefinierter Speicherzugriff)
- Sprachkonstrukte:
  - **throw**(x) (mit $x \in X$) wirft Ausname x
  - **try** c_1 **catch** $f \to c_2$ fängt Ausnahme oder Laufzeitfehler $f \in X + E$

Erweiterung der Sprache

$$
\begin{aligned}
l &::= i \mid l.i \mid l[e] \\
e &::= \mathbb{Z} \mid \mathit{true} \mid \mathit{false} \mid l \\
&\quad \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2 \\
&\quad \mid e_1 == e_2 \mid e_1 < e_2 \\
&\quad \mid \,!\, e \mid e_1 \,\&\&\, e_2 \mid e_1 \,||\, e_2 \\
&\quad \mid l := e \\
&\quad \mid \textbf{throw}(x)
\end{aligned}
$$

$$
\begin{aligned}
c &::= e \mid \textbf{if}\ (e)\ \textbf{then}\ c_1\ \ \textbf{else}\ \ c_2 \mid \textbf{while}\ (e)\ c \mid c_1; c_2 \mid \textbf{nil} \\
&\quad \mid \textbf{try}\ \ c_1\ \ \textbf{catch}\ \ x \to c_2
\end{aligned}
$$

Semantik

- Für Ausdrücke und Anweisungen:

$$F = X + E$$
$$\Sigma \rightharpoonup (\mathbf{V} + F) \times \Sigma$$
$$\Sigma \rightharpoonup (() + F) \times \Sigma$$

- Reguläre Auswertung (ohne Fehler oder Ausnahmen) gibt Wert in $\mathbf{V}$ bzw. () (dient als "Marker") zurück
- Laufzeitfehler oder Ausnahmen geben $f \in X + E$ zurück.

Regeln I: L-Werte

Es werden nur die **zusätzlichen** Regeln aufgeführt:

$$\frac{\Gamma \vdash \langle l, \sigma\rangle \to_{Lexp} \langle f, \sigma'\rangle \qquad f \in X + E}{\Gamma \vdash \langle l.i, \sigma\rangle \to_{Lexp} \langle f, \sigma'\rangle}$$

$$\frac{\Gamma \vdash \langle l, \sigma\rangle \to_{Lexp} \langle f, \sigma'\rangle \qquad f \in X + E}{\Gamma \vdash \langle l[e], \sigma\rangle \to_{Lexp} \langle f, \sigma'\rangle}$$

$$\frac{\Gamma \vdash \langle e, \sigma\rangle \to_{Exp} \langle f, \sigma'\rangle \qquad f \in X + E}{\Gamma \vdash \langle l[e], \sigma\rangle \to_{Lexp} \langle f, \sigma'\rangle}$$

$$\frac{\Gamma \vdash l : \textbf{int} \text{ or } \Gamma \vdash l : \textbf{bool} \qquad \Gamma \vdash \langle l, \sigma\rangle \to_{Lexp} \langle n, \sigma'\rangle \qquad n \notin \mathrm{dom}(\sigma')}{\Gamma \vdash \langle l, \sigma\rangle \to_{Exp} \langle E_1, \sigma'\rangle}$$

Regeln II: Ausdrücke

Es werden nur **zusätzliche** Regeln aufgeführt:

$$\frac{\Gamma \vdash \langle e_1, \sigma\rangle \to_{Exp} \langle f, \sigma'\rangle \qquad f \in X + E}{\Gamma \vdash \langle e_1 / e_2, \sigma\rangle \to_{Exp} \langle f, \sigma'\rangle}$$

$$\frac{\Gamma \vdash \langle e_1, \sigma\rangle \to_{Exp} \langle v, \sigma'\rangle, v \in \mathbb{Z} \qquad \Gamma \vdash \langle e_2, \sigma'\rangle \to_{Exp} \langle f, \sigma''\rangle \qquad f \in X + E}{\Gamma \vdash \langle e_1 / e_2, \sigma\rangle \to_{Exp} \langle f, \sigma''\rangle}$$

$$\frac{\Gamma \vdash \langle e_1, \sigma\rangle \to_{Exp} \langle v, \sigma'\rangle, v \in \mathbb{Z} \qquad \Gamma \vdash \langle e_2, \sigma'\rangle \to_{Exp} \langle 0, \sigma''\rangle}{\Gamma \vdash \langle e_1 / e_2, \sigma\rangle \to_{Exp} \langle E_0, \sigma''\rangle}$$

$$\frac{\Gamma \vdash \langle e_1, \sigma\rangle \to_{Exp} \langle v_1, \sigma'\rangle, v_1 \in \mathbb{Z}, \qquad \Gamma \vdash \langle e_2, \sigma'\rangle \to_{Exp} \langle v_2, \sigma''\rangle, v_2 \in \mathbb{Z}, v_2 \neq 0}{\Gamma \vdash \langle e_1 / e_2, \sigma\rangle \to_{Exp} \langle v_1 \div v_2, \sigma''\rangle}$$

Regeln III: Anweisungen

Es werden nur **zusätzliche** Regeln aufgeführt:

$$\frac{\Gamma \vdash \langle c_1, \sigma\rangle \to_{Stmt} \langle (), \sigma'\rangle}{\Gamma \vdash \langle \textbf{try}\ \ c_1\ \ \textbf{catch}\ \ x \to c_2, \sigma\rangle \to_{Stmt} \langle (), \sigma'\rangle}$$

$$\frac{\Gamma \vdash \langle c_1, \sigma\rangle \to_{Stmt} \langle e_1, \sigma'\rangle \qquad e_1 \neq e_2}{\Gamma \vdash \langle \textbf{try}\ \ c_1\ \ \textbf{catch}\ \ e_2 \to c_2, \sigma\rangle \to_{Stmt} \langle e_1, \sigma'\rangle}$$

$$\frac{\Gamma \vdash \langle c_1, \sigma\rangle \to_{Stmt} \langle f, \sigma'\rangle \qquad \Gamma \vdash \langle c_2, \sigma'\rangle \to_{Stmt} \langle r, \sigma''\rangle}{\Gamma \vdash \langle \textbf{try}\ \ c_1\ \ \textbf{catch}\ \ f \to c_2, \sigma\rangle \to_{Stmt} \langle r, \sigma''\rangle}$$

Zusammenfassung

- Ausnahmen: reglementierte Sprünge
  - In Java, Python, Haskell recht ähnlich
  - In C nicht vorhanden
  - Sollten **unerwartete** und **seltene** Situationen behandeln