Reaktive Programmierung
Vorlesung 16 vom 10.07.2019
Theorie der Nebenläufigkeit

Christoph Lüth, Martin Ring

Universität Bremen

Sommersemester 2019

Fahrplan

- Einführung
- Monaden und Monadentransformer
- Nebenläufigkeit: Futures and Promises
- Aktoren I: Grundlagen
- Aktoren II: Implementation
- Meta-Programmierung
- Bidirektionale Programmierung
- Reaktive Ströme I
- Reaktive Ströme II
- Funktional-Reaktive Programmierung
- Software Transactional Memory
- Eventual Consistency
- Robustheit und Entwurfsmuster
- Theorie der Nebenläufigkeit, Abschluss

Theorie der Nebenläufigkeit

- Nebenläufige Systeme sind **kompliziert**
  - Nicht-deterministisches Verhalten
  - Neue Fehlerquellen wie **Deadlocks**
  - Schwer zu testen
- Reaktive Programmierung kann diese Fehlerquellen **einhegen**
- **Theoretische Grundlagen** zur Modellierung nebenläufiger Systeme
  - zur **Spezifikation** (CSP)
  - aber auch als **Berechnungsmodell** (π-Kalkül)

Temporale Logik, Prozessalgebren und Modelchecking

- Prozessalgebren und temporale Logik beschreiben **Systeme** anhand ihrer **Zustandsübergänge**
- Ein System ist dabei im wesentlichen eine **endliche Zustandsmaschine** $\mathcal{M} = \langle S, \Sigma, \rightarrow \rangle$ mit Zustandsübergang $\rightarrow \subseteq S \times \Sigma \times S$
- Temporale Logiken reden über **eine** Zustandsmaschine
- Prozessalgebren erlauben **mehrere** Zustandsmaschinen und ihre **Synchronisation**
- Der Trick ist **Abstraktion**: mehrere interne Zustandsübergänge werden zu einem Zustandsübergang zusammengefaßt

Einfache Beispiele

- Einfacher Kaffee-Automat:

$$P = 10c \rightarrow coffee \rightarrow P$$

- Kaffee-Automat mit Auswahl:

$$P = 10c \rightarrow coffee \rightarrow P \;\Box\; 20c \rightarrow latte \rightarrow P$$

- Pufferprozess:

$$COPY = left?x \rightarrow right!x \rightarrow COPY$$

NB. Eingabe ($c?x$) und Ausgabe ($c!x$) sind **reine Konvention**.

CSP: Syntax

Gegeben Prozeßalphabet Σ, besondere Ereignisse $\checkmark, \tau$

$P ::= Stop \mid a \to P \mid \mu\, P.F(P)$	fundamentale Operationen
$\mid P \,\Box\, Q \mid P \sqcap Q$	externe und interne Auswahl
$\mid P \parallel Q \mid P \underset{X}{\parallel} Q$	synchronisiert parallel
$\mid P \,\vert\vert\vert\, Q$	unsynchronisiert parallel
$\mid P \setminus X$	hiding
$\mid Skip \mid P;\ Q$	sequentielle Komposition

Externe vs. interne Auswahl

- Interne Zustandsübergänge (τ) sind **nicht beobachtbar**, aber können Effekte haben.
- Vergleiche:

$$a \to b \to \mathit{Stop} \,\Box\, a \to c \to \mathit{Stop}$$

$$a \to b \to \mathit{Stop} \,\sqcap\, a \to c \to \mathit{Stop}$$

$$a \to (b \to \mathit{Stop} \,\Box\, c \to \mathit{Stop})$$

$$a \to (b \to \mathit{Stop} \,\sqcap\, c \to \mathit{Stop})$$

Beispiel: ein Flugbuchungssystem

- Operationen des Servers:
  - Nimmt Anfragen an, schickt Resultate (mit flid)
  - Nimmt Buchungsanfragen an, schickt Bestätigung (ok) oder Fehler (fail)
  - Nimmt Stornierung an, schickt Bestätigung
- Unterschied zwischen **interner** Auswahl $\sqcap$ (Server trifft Entscheidung), und **externer** Auswahl $\Box$ (Server reagiert)

$$
\begin{aligned}
SERVER = \; & query?(from, to) \rightarrow result!flid \rightarrow SERVER \\
& \Box\, booking?flid \rightarrow (ok \rightarrow SERVER \sqcap fail \rightarrow SERVER) \\
& \Box\, cancel?flid \rightarrow ok \rightarrow SERVER
\end{aligned}
$$

Eingabe $(c?x)$ und Ausgabe $(c!a)$ sind reine Konvention

Beispiel: ein Flugbuchungssystem

- Operationen des Servers:
  - Nimmt Anfragen an, schickt Resultate (mit flid)
  - Nimmt Buchungsanfragen an, schickt Bestätigung (ok) oder Fehler (fail)
  - Nimmt Stornierung an, schickt Bestätigung
- Unterschied zwischen **interner** Auswahl $\sqcap$ (Server trifft Entscheidung), und **externer** Auswahl $\Box$ (Server reagiert)

$$\begin{aligned} SERVER = \; & query \rightarrow result \rightarrow SERVER \\ & \Box\, booking \rightarrow (ok \rightarrow SERVER \sqcap fail \rightarrow SERVER) \\ & \Box\, cancel \rightarrow ok \rightarrow SERVER \end{aligned}$$

Eingabe $(c?x)$ und Ausgabe $(c!a)$ sind reine Konvention

Beispiel: ein Flugbuchungssystem

- Der Client:
  - Stellt Anfrage
  - wenn der Flug richtig ist, wird er gebucht;
  - oder es wird eine neue Anfrage gestellt.

$$CLIENT = query \rightarrow result \rightarrow (booking \rightarrow ok \rightarrow CLIENT \sqcap CLIENT)$$

- Das Gesamtsystem — Client und Server **synchronisiert**:

$$SYSTEM = CLIENT \parallel SERVER$$

Beispiel: ein Flugbuchungssystem

- Der Client:
  - Stellt Anfrage
  - wenn der Flug richtig ist, wird er gebucht;
  - oder es wird eine neue Anfrage gestellt.

$$CLIENT = query \rightarrow result \rightarrow (booking \rightarrow ok \rightarrow CLIENT \sqcap CLIENT)$$

- Das Gesamtsystem — Client und Server **synchronisiert**:

$$SYSTEM = CLIENT \parallel SERVER$$

- Problem: **Deadlock**
  - Es gibt **Werkzeuge** (Modelchecker, z.B. FDR), um solche Deadlocks in Spezifikationen zu finden

Beispiel: ein Flugbuchungssystem

- Der Client:
  - Stellt Anfrage
  - wenn der Flug richtig ist, wird er gebucht;
  - oder es wird eine neue Anfrage gestellt.

$$\begin{aligned} CLIENT \quad = query \rightarrow result \rightarrow \\ (booking \rightarrow \quad (ok \rightarrow CLIENT \\ \qquad\qquad\qquad \Box\, fail \rightarrow CLIENT) \\ \qquad \sqcap\, CLIENT) \end{aligned}$$

- Das Gesamtsystem — Client und Server **synchronisiert**:

$$SYSTEM = CLIENT \parallel SERVER$$

- Problem: **Deadlock**
  - Es gibt **Werkzeuge** (Modelchecker, z.B. FDR), um solche Deadlocks in Spezifikationen zu finden

Ziele der Semantik von Prozesskalkülen

- **Reasoning about processes by their external behaviour**
- Untersuchung von
  - Verfeinerung (Implementation)
  - **deadlock:** Keine Transition möglich
  - **livelock:** Divergenz
- Grundlegender Begriff: **Äquivalenz (Gleichheit) von Prozessen**

Operationale Semantik für CSP (I)

Definition: Labelled Transition System (LTS)

Ein **labelled transition system (LTS)** ist $L = (N, A, \rightarrow)$ mit Menge N der Knoten (Zustände), Menge A von Labels und Relation $\{\xrightarrow{a} \subseteq N \times N\}_{a \in A}$ von Kanten (Zustandsübergänge).

Hier: $N = P$,$A = \Sigma \bigcup \{\checkmark, \tau\}$, $\rightarrow$ definiert wie folgt:

$$\frac{}{e \rightarrow P \xrightarrow{a} P[a/e]} \; a \in comms(e)$$

$$\frac{}{P \sqcap Q \xrightarrow{\tau} P} \qquad \frac{}{P \sqcap Q \xrightarrow{\tau} Q}$$

Operationale Semantik für CSP (II)

$$\frac{P \xrightarrow{\tau} P'}{P \,\Box\, Q \xrightarrow{\tau} P' \,\Box\, Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P \,\Box\, Q \xrightarrow{\tau} P \,\Box\, Q'}$$

$$\frac{P \xrightarrow{a} P'}{P \,\Box\, Q \xrightarrow{a} P'} \; a \neq \tau \qquad \frac{Q \xrightarrow{a} Q'}{P \,\Box\, Q \xrightarrow{a} Q'} \; a \neq \tau$$

$$\frac{P \xrightarrow{x} P'}{P \setminus B \xrightarrow{\tau} P'} \; x \in B \qquad \frac{P \xrightarrow{x} P'}{P \setminus B \xrightarrow{x} P' \setminus B} \; x \notin B$$

Operationale Semantik für CSP (III)

$$\frac{P \xrightarrow{\tau} P'}{P \underset{X}{\parallel} Q \xrightarrow{\tau} P' \underset{X}{\parallel} Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P \underset{X}{\parallel} Q \xrightarrow{\tau} P \underset{X}{\parallel} Q'}$$

$$\frac{P \xrightarrow{a} P'}{P \underset{X}{\parallel} Q \xrightarrow{a} P' \underset{X}{\parallel} Q} \; a \notin X \qquad \frac{Q \xrightarrow{a} Q'}{P \underset{X}{\parallel} Q \xrightarrow{a} P \underset{X}{\parallel} Q'} \; a \notin X$$

$$\frac{P \xrightarrow{a} P' \; Q \xrightarrow{a} Q'}{P \underset{X}{\parallel} Q \xrightarrow{a} P' \underset{X}{\parallel} Q'} \; a \in X$$

Denotationale Semantik für CSP

- **Operationale** Semantik erklärt das **Verhalten**, erlaubt kein **Reasoning**
- **Denotationale** Semantik erlaubt **Abstraktion** über dem Verhalten
- Für CSP: Denotat eines Prozesses ist:
  - die Menge aller seiner **Traces**
  - die Menge seiner **Traces** und **Acceptance-Mengen**
  - die Menge seiner **Traces** und seiner **Failure/Divergence**-Mengen

Anwendungsgebiete für CSP

- Modellierung nebenläufiger Systeme (Bsp: ISS)
- Verteilte Systeme und verteilte Daten
- Analyse von Krypto-Protokollen
- Hautpwerkzeug: der Modellchecker **FDR**
  - `http://www.cs.ox.ac.uk/projects/fdr/`