Formale Modellierung
Vorlesung 13 vom 01.07.13: Rückblick und Ausblick

Serge Autexier & Christoph Lüth

Universität Bremen

Sommersemester 2013

Fahrplan

- Teil I: Formale Logik
- Teil II: Spezifikation und Verifikation
- Teil III: Schluß
  - Rückblich & Ausblick

Das Tagesmenü

- Zur Abrundung: JML
- Rückblick
- Ausblick

JML

Java Modeling Language (JML)

- Zentral: funktionale Korrektheit
- "Design by contract"
- Spezifikation nahe am Code (Annotationen)
- Vor/Nachbedingungen, Invarianten
- Werkzeuge: ESC/Java2, Mobius

JML: Erstes Beispiel

```
public abstract class LinearSearch
{
 //@ requires j >= 0;
 public abstract /*@ pure @*/ boolean f(int j);

 //@ ensures 0 <= \result;
 //@ ensures (\exists int j; 0 <= j && j <= \result; f(j));
 public abstract /*@ pure @*/ int limit();

 /*@ public normal_behavior
   @   requires (\exists int i; 0 <= i && i <= limit(); f(i));
   @   assignable \nothing;
   @   ensures f(\result) &&
   @           (\forall int i; 0<= i && i < \result; ! f(i));
   @*/
 public int find()
}
```

JML: Wichtigste Schlüsselworte

- Für Klassen und Interfaces:
  - `invariant`
  - `spec_public`
  - `nullable`
- Für Methoden und Konstruktoren:
  - `requires`
  - `ensures`
  - `assignable`
  - `pure`
- Im Code:
- `assert`
- `invariant`, `assignable`

JML: Längeres Beispiel

Printed by Christoph Lueth

Jun 30, 13 22:18 **BoundedStack.java** Page 1/1

```
public class BoundedStack {

  private /*@ spec_public nullable @*/
     Object[] elems;
  private /*@ spec_public @*/ int size = 0;

  //@ public invariant 0 <= size;
  /*@ public invariant elems != null
    @   && (\forall int i;
    @           size <= i && i < elems.length;
    @           elems[i] == null);
    @*/

  /*@ requires 0 < n;
    @ assignable elems;
    @ ensures elems.length == n;
    @*/
  public BoundedStack(int n) {
    elems = new Object[n];
  }

  /*@ requires size < elems.length-1;
    @ assignable elems[size], size;
    @ ensures size == \old(size+1);
    @ ensures elems[size-1] == x;
    @ ensures_redundantly
    @     (\forall int i; 0 <= i && i < size-1;
    @               elems[i] == \old(elems[i]));
    @*/
  public void push(Object x) {
    elems[size] = x;
    size++;
  }

  /*@ requires 0 < size;
    @ assignable size, elems[size-1];
    @ ensures size == \old(size-1);
    @ ensures_redundantly
    @      elems[size] == null
    @   && (\forall int i; 0 <= i && i < size-1;
    @               elems[i] == \old(elems[i]));
    @*/
  public void pop() {
    size--;
    elems[size] = null;
  }

  /*@ requires 0 < size;
    @ assignable \nothing;
    @ ensures \result == elems[size-1];
    @*/
  public /*@ pure @*/ Object top() {
    return elems[size-1];
  }
}
```

Sunday June 30, 2013 1/1

Zusammenfassung JML

- Modellierung der Daten durch Java
- Zusätzliche Annotation der Programme mit Korrektheitsbedingungen
- Dadurch leichtgewichtig — erleichtert Einführung
- Benefits:
  - Generierung von Testfällen
  - Statische Analyse (“erweiterte Typprüfung”)
  - Korrektheitsbeweis (automatisch oder interaktiv)
- Nachteile:
  - Umständliche Modellierung — kein Abstraktionsgewinn

Rückblick

Logiken und Spezifikationsformalismen

	Datenmodellierung	Programmbegriff
Isabelle/HOL	induktive Datentypen	totale rekursive Funktionen
Z	(induktive) Mengen	Vor/Nachzustand[1]
UML	Klassendiagramme	OCL
JML	Java	Vor/Nachzustand, Invarianten

[1] Invarianten durch Ξ

Unsere Reise durch die Logik

	Entscheidbar?	Vollständig?	Werkzeuge (Beweiser)
Aussagenlogik	J	J	SAT-Solver
Presburger	J	J	SMT-Solver: Z3, CVC
Peano-Ar.	N	J	
FOL	N	J	ATPs: SPASS, Vampire
FOL + Induktion	N	N	KIV, KeY, Inka
HOL	N	N	ITPs: Isabelle, Coq, HOL, PVS

Ausblick

Wohin von hier?

- Verifikation von Hardware und Software
  - Formalisierung von Hardware oder Software (Programmiersprache)
  - Nachweis der Eigenschaften wird Beweis
- Weitere Ausdrucksmächtigkeit:
  - Nebenläufigkeit: Prozesskalküle (CSP), Modallogik
  - Zeitliche Aspekte (Temporallogik)

Prozesskalküle (Prozessalgebren)

- Modellierung nebenläufige Systeme
- Werkzeugunterstützung: Modelchecker FDR
- Beispiel (CSP): ein Flugbuchungssystem

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

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

$$SYSTEM = CLIENT \parallel SERVER$$

Prozesskalküle (Prozessalgebren)

- Modellierung nebenläufige Systeme
- Werkzeugunterstützung: Modelchecker FDR
- Beispiel (CSP): ein Flugbuchungssystem

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

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

$$SYSTEM = CLIENT \parallel SERVER$$

Problem: Deadlock

Prozesskalküle (Prozessalgebren)

- Modellierung nebenläufige Systeme
- Werkzeugunterstützung: Modelchecker FDR
- Beispiel (CSP): ein Flugbuchungssystem

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

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

$$SYSTEM = CLIENT \parallel SERVER$$

Temporale Logiken

- System wird beschrieben als FSM
  - Ggf. mit Übergangszeiten
  - Graphische Notation oder DSL (Promela für SPIN)
- Spezifikation als temporallogische Formel:
  - "Irgendwann muss ein Event HALT (H) auftreten": $\mathbf{F}H$
  - "Es darf nie ein STOP (S) ohne eine WARNUNG (W) zuvor auftreten":

$$\mathbf{G}((W \longrightarrow \mathbf{F}S) \vee \neg S)$$

- Lineare vs. baumartige Zeit (LTL, CTL), TLA
- Werkzeuge: SPIN, UPPAAL

Fazit

Dafür und Dagegen

Was spricht für formale Modellierung in der Entwicklung?

- Anforderungen werden eindeutig formuliert
- Randbedingungen werden seltener (gar nicht) übersehen
- Fehler werden früher gefunden
- Werkzeugunterstützung möglich
  - Von erweitertem Typcheck bis automatischen/interaktiven Beweis
- Normen (IEC 61508:3, CENELEC EN50128, DO 178B) fordern formale Methoden für hohe Sicherheitsstufen (SIL 3, Level B)

Dafür und Dagegen

Was spricht für formale Modellierung in der Entwicklung?

- Anforderungen werden eindeutig formuliert
- Randbedingungen werden seltener (gar nicht) übersehen
- Fehler werden früher gefunden
- Werkzeugunterstützung möglich
  - Von erweitertem Typcheck bis automatischen/interaktiven Beweis
- Normen (IEC 61508:3, CENELEC EN50128, DO 178B) fordern formale Methoden für hohe Sicherheitsstufen (SIL 3, Level B)

Was spricht gegen formale Modelierung?

- Höherer Zeitaufwand
- Qualifikation des Personals
- Verlust an Agilität

Zusammenfassung

Formale Modellierung

Beschreibung der Welt durch Mittel der mathematischen Logik

- Beispiele: HOL, die Z Notation, UML/OCL, JML
- Vorteile:
  - Spezifikationen eindeutig formuliert
  - Nachweis von Eigenschaften möglich
  - Formale Verifikation möglich