Formale Methoden

Author:

Sarah Schröcker

27. Mai 2015

1 Einführung

Formale Methoden ist nicht nur in der Theorie, sondern vor allem auch in der Praxis sehr wichtig. Die Software einer Rakete, eines Herzschrittmachers oder einer Aktienverwaltung muss korrekt funktionieren, um wirtschaftliche und Menschen betreffende Schäden auszuschließen . Damit diese Korrektheit nachgewiesen werden kann, spielen Formale Methoden eine zentrale Rolle.

In diesem Artikel wird ein Überblick geschaffen, wie wichtig Korrektheit in der Software ist und was geschehen kann wenn richtiges Testen nicht ernst genommen oder ignoriert wird.

2 Beweissysteme

Es kann ziemlich mühselig sein Korrektheit von Programmen zu beweisen. Deshalb gibt es Beweissysteme, bei denen Regeln angegeben werden, um dies zu vereinfachen. Dieser Ansatz wurde von dem britischen Informatiker C.A.R. Hoar entwickelt. Seine Entwicklung wird auch Hoare-Kalkül genannt, welches im nächsten Abschnitt beschrieben wird.

2.1 Hoare Kalkül

Der Hoare-Kalkül dient zum Beweisen von partieller und totaler Korrektheit. Das Hoare-Tripel {Q} P {R} beschreibt, wie ein Programmteil den Zustand einer Berechnung verändert. Dabei sind P und Q Zusicherungen und S ist ein Programmsegment. Wenn die Vorbedingung (P) zutrifft, gilt nach der Ausführung des Programmsegements (S) die Nachbedingung (Q).

Auf folgender Abbildung 1 finden wir die Regeln des Hoare Kalkül. Für mehr Informationen dazu verweise ich auf den Foliensatz von G. Moser[3].

$$Zuweisung: \frac{}{\{Q\{x \mapsto t\}\}x := t\{Q\}}$$

$$Komposition: \frac{\{Q\}P1\{R\} \quad \{R\}P2\{S\}}{\{Q\}P1; P2\{S\}}$$

$$Konsequenz: \frac{\{Q'\}P\{R'\}}{\{Q\}P\{R\}} Q \models Q', R' \models R$$

$$Iteration: \frac{\{I \wedge B\}P\{I\}}{\{I\}while\ B\ do\ P\ end\{I \wedge \neg B\}}$$

Abbildung 1: Regeln nach Tony Hoare

Es gibt 2 Interpretationen der Bedeutung von Hoare-Formeln: Die partielle und die totale Korrektheit.
Partielle Korrektheit:
Wenn P im Anfangszustand von S gilt und wenn S terminiert, dann gilt Q nach Ausführung von S[2].
Totale Korrektheit:
-Wenn P im Anfangszustand von S gilt, dann terminiert S und gilt Q nach Ausführung von S.
-Totale Korrektheit = Partielle Korrektheit + Terminierung[2]

3 Assertions

Mit Zusicherungen kann man überprüfen, ob an einer Stelle des Programms eine Eigenschaft (d.h. ein Boolescher Ausdruck) gültig ist. Die Überprüfung einer Bedingung erfolgt mit assert. Ist dieser Ausdruck false, tritt ein Fehler vom Typ AssertionError auf. Die Gültigkeit von Zusicherungen kann man mit dem Hoare-Kalkül beweisen[1].
Nachteile:
- Programmabbrüche durch gescheiterte assert-Befehle nicht akzeptabel
- Prüfung kostet Laufzeit. Problem bei zeitkritischen Anwendungen

Lösung: Bei den Compilern einstellen, ob assert-Befehle mit kompiliert werden sollen oder nicht. Für Laufzeitmessungen oder Auslieferungen an den Kunden werden assert-Befehle nicht übersetzt.

Das Programm aus Listing 1 berechnet in p die j-fache Potenz von i. Die Korrektheit wird von einem assert-Befehl überprüft.

Listing 1: Java-Programm mit assert-Befehl

```
int i = 7;
int j = 4;
int p = 1;
int k = 0;
while (k < j) {
        p *= i;
        k++;
} assert(k == j && p == Math.pow(i,k));
```

Es gibt 2 Ansätze aus dem Bereich der SW-Tests, die gewährleisten, dass diese auch getestet werden:
Erster Ansatz: Bildung so genannter Äquivalenzklassen
Für jede Äquivalenzklasse muss es einen Testfall geben.
Zweiter Ansatz: Anweisungs- oder Knotenüberdeckung
Die Testfälle werden so geschrieben, so dass bei der Ausführung aller Testfälle jede Anweisung mindestens einmal durchgeführt wird.

Im Bereich der objektorientierten Sprachen haben sich Unit-Test für die Ausführung und Überwachung der Tests gut durchgesetzt. In Eclipse ist JUnit bereits integriert. Mehr Informationen dazu sind in den Foliensätzen von E. Zangerle zu finden[5].

4 Beispiele für Softwarefehler

Trotz der rasanten Entwicklung der Softwaretechnik ist die Entwicklung fehlerfreier Software immer noch eine große Herausforderung. In der folgenden Tabelle sind ein paar Beispiele dazu aufgelistet.

1962	Mariner 1, Cape Canevaral, Trägerrakete sprengt sich nach Kursabweichung selbst, Grund für die Abweichung war ein Komma statt eines Punktes im Quellcode.
1971	Eole1, französischer Satellit zur Koordination von 141 Wetterballons; 72 Ballons erhalten Befehl vom Satelliten, Daten zu senden, interpretieren dies aber als Befehl zur Selbstzerstörung.
1994	Software-Fehlverhalten ist mitverantwortlich für den Absturz eines A330 bei Toulouse. 7 Tote.
1999	Mars Polar Lander; Landefahrzeug stürzt aus 40 Metern Höhe auf die Oberfläche; drei Sensoren hatten die Erschütterung beim Ausfahren der Landebeine als Bodenkontakt interpretiert.

2003	Flugabwehrsystem Patriot identifiziert in Kuwait befreundetes britisches Kampfflugzeug als Feind und schießt es ab.
2005	Fehler in der Software zur Berechnung des Arbeitslosengeldes 2 der Bundesagentur für Arbeit, Auszahlungen fanden nicht statt, da bei neunstelligen Kontonummern am Ende eine Null ergänzt wurde, um auf das gewünschte zehnstellige Format zu kommen.
2005	Durch einen Software-Fehler bei der Weiterleitung von E-Mails eines Providers wird eine E-Mail eines Abgeordneten des sächsischen Landtages mehrere Tausend Mal an die Empfänger zugestellt

Tabelle 1: Liste einiger Softwarefehler[1] [4]

Diese Beispiele zeigen zum einen typische Anwendungsbereiche formaler Methoden, zum anderen die praktische, gesellschaftliche und wirtschaftliche Relevanz des Problems.

5 Schlussfolgerung

Vor allem in der Praxis sind Formale Methoden von großer Bedeutung. In der Software kann es durch Spezifikations-, Design- und Programmierfehlern zu erheblichen Risiken in technischen Anwendungen kommen. Durch die in diesem Artikel vorgestellten Methoden, kann der Risikofaktor reduziert werden.

Literatur

[1] S. Kleuker: *Formale Modelle der Softwareentwicklung.*, Vieweg+Teubner, 1. Auflage, 2009

[2] Ruth Breu: *Softwareentwicklung und Projektmanagement*, Vorlesungsfolien SS15, Universität Innsbruck

[3] G.Moser: *Einführung in die Theoretische Informatik*, 3. Auflage WS13

[4] Wolfgang Reif: *Formale Methoden für sicherheitskritische Software – Der KIV-Ansatz*

[5] E. Zangerle: http://dbis-informatik.uibk.ac.at/files/ext/lehre/ss14/ProgMeth/VO/PM-10-JUnit.pdf