Vorlesung
Programmverifikation

Prof. Dr. Uwe Petermann
FB IMN HTWK Leipzig 04251 Leipzig (Germany)
uwe@imn.htwk-leipzig.de

13. Juni 2006

Zusammenfassung

Software ist zum Herzstück vieler technischer Systeme geworden. Dadurch werden Fragen der Zuverlässigkeit technischer Systeme immer mehr zu Fragen der Korrektheit von Software. Spektakuläre Computerpannen der letzten Zeit bestätigen das eindrucksvoll. Weil das durch fehlerhafte Software verursachte Schadensrisiko oftmals unkalkulierbar ist, entsteht ein zunehmender ökonomischer Zwang, korrekte Software zu benutzen. Produzenten, die solche Software herstellen können, werden einen Wettbewerbsvorteil erlangen.

Den derzeit besten Schutz vor Software-Fehlern bieten formale Spezifikation und Verifikation (vgl. Wolfgang Reif, FMG Uni Augsburg: Risikofaktor Software).

Intelligente Werkzeuge zur Unterstützung dieser Methoden ermöglichen heute deren praktischen Einsatz. Mit Hilfe von Verifikationssystemen kann die Korrektheit eines Programms bezüglich seiner Spezifikation nachgewiesen werden. Erfolgreiche Systeme (z.B. KIV) nutzen Spezifikationssprachen, um hierarchisch strukturierte Softwaresysteme zu beschreiben.

Die Vorlesung behandelt die Methoden der Spezifikation und Verifikation von informationstechnischen Systemen. Die Methoden werden durch Spezifikations- und Verifikationsexperimente, die mit dem Verifikationssystem KIV bearbeitet werden, illustriert.

Dieses Dokument als .pdf-Datei

Überblick zu Vorlesung ProgrammVerifikation

Ablauf

Vorlesung (2SWS) und Übung/Rechnerübung (2SWS)

Teilnahme

Teilnahme an den zur Vorlesung gehörenden Lehrveranstaltungen ist Pflicht

Leistungsnachweis

Prüfungsrelevante Studienleistung Projekt: Februar 2006

Zum Gegenstand:

Form der Lehrveranstaltung:

- Vorlesung (2SWS) und Übung/Rechnerübung (2SWS)
- Teilnahme an den Lehrveranstaltungen ist Pflicht
- Leistungsnachweis: Prüfungsrelevante Studienleistung
  Einzelheiten siehe unten

Mitschriften und andere Materialien zur Vorlesung

Einführung und erstes Projekt

Einführung und erstes Projekt:

Prädikatenlogik als Sprache formaler Spezifikationen

Prädikatenlogik als Sprache formaler Spezifikationen:

Formulieren und Nachweisen von Eigenschaften

Formulieren und Nachweisen von Eigenschaften:

Spezifikation von Datenstrukturen

Spezifikation von Datenstrukturen:

Spezifikationsmethodik

Spezifikationsmethodik:

Implementierungen

Implementierung von Spezifikationen und deren Korrektheit:

Beweisen

Nachweis von Eigenschaften mittels Prädikatenlogik und Induktion:

Automatisierung

Vereinfachungsregeln und Heuristiken:

Nachweis der Korrektheit von Implementierungen

Korrektheitsnachweis für Implementierungen:

Übungen

- Hinweise für die Konfiguration:
- KW 41/42:
  - Übung:
  - Projekt:
- KW 43/44:
- KW 45/46:
- KW 47/48:
- KW 51:
- KW 02/03:

Themenvorschläge für die prüfungsrelevante Studienlesitung

Folgende Themen sind als Beispiele zu betrachten. Eigene Ideen können realisiert werden.

Jedoch ist **in jedem Fall** eine sorgfältige Abstimmung mit dem Lesenden nötig.

Klassische Datenstrukturen und Algorithmen

1. Implementierung von Datenstrukturen I

   Implementierung von Graphen (vgl. Spezifikation `digraph` in Bibliothek `lib-digraph`) durch Adjazenzmatrizen.

2. Implementierung von Datenstrukturen II

   Implementierung von Wörterbüchern (vgl. Spezifikation `store-xyz` in Bibliothek `lib-store`).

3. Spezifikation und Implementierung eines Mergesort-Algorithmus. Abarbeitungsbeispiel:

   11 . 7 . 5 . 9 . 11 . 4 . 15 . 10 . 17

   7 11 . 5 9 . 4 11 . 10 15 . 17

   5 7 9 11 . 4 10 11 15 . 17

   4 5 7 9 10 11 11 15 . 17

4. Implementierung von Mengen durch duplikatfreie sortierte Listen.

5. Implementierung und Verifikation von Algorithmen, die Pixelmuster einfacher geometrischer Figuren erkennen (z.B. Rechtecke, Dreiecke, ...).

6. andere Algorithmen und Datenstrukturen.

   Wichtig: Es genügt nicht, den *Ablauf* eines Algorithmus kennen! Sie müssen verstehen, welche Invarianten die im Algorithmus vorkommenden Iterationen bzw. Rekursionen beschreiben. Ersteres findet man in den meisten Büchern. Letzteres oft nur unvollständig.

Kommunikationsprotokolle

1. Spezifikation und Implementierung eines Spam-Filters

   Die Wichtigkeit von Spam-Filtern ist unbestritten.

   Die Aufgabe besteht darin, aus einem Mail-File, wie es in UNIX-Systemen üblich ist, die Spam-Mails zu entfernen.

   Spezifikation und Implementierung sollten so erfolgen, daß man sich die Realisierung in einem tatsächlichen System sowohl als Bearbeitung ("Offline") einer Datei als auch als Bearbeitung eines Stroms ("Online" oder "on the fly") vorstellen kann.

   Als Spam sollen Mails aufgefaßt werden, die bestimmte Kriterien erfüllen. Eine einfache Version könnte untersuchen, ob Wörter aus einer vorgegebenen Liste im Betreff-Feld vorkommen oder ob die Absenderadresse einer schwarzen Liste engehört.

   Hinweis:

   Export: Mail-File als Folge von Mails auffassen. Mails als (generische ?) Datensatzspezifikation darstellen

   Import: Mail-File als Folge von Zeilen (Zeichenketten) auffassen.
   D.h. Implementierung ist abstrakt. Argumentieren, wie nachvollziehbarer Zusammenhang mit einer tatsächlichen Implementierung hergestellt werden kann.

2. Beschreiben Sie ein Prokoll für den Internethandel

   (a) Protokoll

      i. Händler: Angebot (Ware, Preis, Zahlungsmodalität)
      ii. Kunde: Annahme
      iii. Bezahlung
      iv. Lieferung

   (b) Teilaufgaben

i. Formalisierung des Protokolls als Folge von Nachrichten in einem Nachrichtenstrom (Listen von Nachrichten)
ii. Formulieren von Sicherheitseigenschaften, die die Beteiligten erwarten.
iii. Prüfen, ob Sicherheitseigenschaften erfüllt sind

3. Formalisieren und Validieren eines Protokolls für den Zugriff auf ein VPN (Virtual Private Network).

   Versionen

   (a) ohne asymmetrische Verschlüsselung
   (b) mit asymmetrischer Verschlüsselung (vgl. server/SSO/Hildmann)

4. andere Protokolle

Weitere Themen

1. Bezahlautomat / Geldwechsler (1 - 2 Themen)

   Es sind die Bezahlfunktion inkl. der Herausgeber von Wechselgeld bei Bezahlautomaten zu spezifizieren, zu implementieren und zu verifizieren.

   In einer Variante könnten auch Geldwechselautomaten betrachtet werden.

   Es sollte in jedem Fall auf eine möglichst generische Lösung geachtet werden.

2. APDU-Tool: Operationen mit Bytearrays und Zeichenketten

   Bei der Programmierung von SmartCard-Applikationen sind Spezifikation und Implementierung der Kommunikation zwischen Lesegerät und SmartCard sicherheitskritisch und zugleich fehlerträchtig. Die Kommunikation erfolgt über Byte-Arrays.

   Als Minimum müssen Operationen für Byte-Arrays und deren Konversion in/aus andern Datentypen, z.B. Zeichenketten, spezifiziert, implementiert und verifiziert werden.

   evtl. tw. in KIV-Smart

3. Flugzeug-Sicherheit: In [2] beschreibt Tim van Beveren Unfälle mit Flugzeugen, die ihre Ursache in fehlerhaften oder mißverstandenen Abbildungen der Steuerung von Flugzeugen beruhten.

   Es soll untersucht werden, ob durch formale Spezifikation und Validierung das Fehlverhalten haette aufgedeckt werden können.

(a) Teilprobleme, deren falsche oder mißverständliche Spezifikation, Implementierung und/oder Dokumentation zu Unfällen führten.
   i. Spoilerextension
   ii. Fahrwerkausfahren

4. Spezifikations-, Implementierungs und Verifikationsaufgaben im Zusammenhang mit **Sicherheitswerkzeugen fuer das Internet**.

5. Spezifikations-, Implementierungs und Verifikationsaufgaben im Zusammenhang mit **Konfigurationswerkzeugen fuer Betriebssysteme**, z.B. der Konfiguration einer Firewall.

   Die Herausforderung besteht darin, Konfigurationswerkzeuge mit dem Anspruch auftreten, dem Anwender behilflich zu sein, ohne ihn mit der vollen Komplexität der zu konfigurierenden Mechanismen zu belasten. In der Praxis ist aber dem Anwender mitunter unklar, wie er die ihm überlassenen Entscheidungen treffen soll. Sind die Werkzeuge fehlerhaft? Ist der Anwender zu dumm? Stellen Konfigurationswerkzeuge einen Versuch der Quadratur des Kreises dar?

6. Genotype - Phenotype:

   Es geht um die Spezifikation von “Chromosomen” und “Individuen” und von deren Zusammenhang im Sinne des Gene Expression Programming (GEP).

   Als Zusatzaufgabe oder in einem verbundenem Projekt könnte die Transformation vom Genotype in den Phenotype implementiert und verifiziert werden.

   Mehr Informationen: In [1]: In GEP (Gene Expression Programming) the individuals are encoded as linear strings of fixed length (the genome or chromosomes) which are afterwards expressed as nonlinear entities of different sizes and shapes (i.e., simple diagram representations or expression trees).

   `http://www.gene-expression-programming.com`

Literatur

[1] C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, pages 87–129, 2001.

[2] T. van Beveren. *Runter kommen sie immer.* Campus, TvBeveren@compuserve.com, 2000.