Korrekte Software: Grundlagen und Methoden
Vorlesung 16 vom 07.07.16: Rückblick & Ausblick

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2016

Fahrplan

- Einführung
- Die Floyd-Hoare-Logik
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Semantiken
- Verifikation: Vorwärts oder Rückwärts?
- Korrektheit des Hoare-Kalküls
- Einführung in Isabelle/HOL
- Weitere Datentypen: Strukturen und Felder
- Funktionen und Prozeduren
- Referenzen und Zeiger
- Frame Conditions & Modification Clauses
- Ausblick und Rückblick

Was gibt's heute?

- Rückblick
- Ausblick
- Feedback

Rückblick

Semantik

- Operational — Auswertungsrelation $\langle c, \sigma \rangle \to \sigma'$
- Denotational — Partielle Funktion $[\![c]\!] : \Sigma \rightharpoonup \Sigma$
- Axiomatisch — Floyd-Hoare-Logik
- Äquivalenz von operationaler und denotationaler Semantik
- Welche Semantik wofür?

Floyd-Hoare-Logik

- Floyd-Hoare-Logik: partiell und total
- $\vdash \{P\}\, c\, \{Q\}$ vs. $\models \{P\}\, c\, \{Q\}$: Vollständigkeit, Korrektheit
- Die sechs Basisregeln
- Zuweisungsregel: vorwärts (Floyd) vs. rückwärts (Hoare)
- Zusammenhang mit denotationaler/operationaler Semantik
- VCG: Schwächste Vorbedingung und stärkste Nachbedingung

Erweiterung der Programmiersprache

- Für jede Erweiterung:
  - Wie modellieren wir semantisch?
  - Wie ändern sich die Regeln der Logik?
- Strukturen und Felder
  - Lokationen, **Lexp**, strukturierte Werte
  - Erweiterte Substitution in Zuweisungsregel
- Prozeduren und Funktionen
  - Modellierung von **return**: Erweiterung zu $\Sigma \rightharpoonup \Sigma \times \mathbf{V}_U$
  - Spezifikation von Funktionen durch Vor-/Nachbedingungen
  - Spezifikation der Funktionen muss im Kontext stehen

Erweiterung der Programmiersprache

- Zeiger und Referenzen
  - Lokationen nicht mehr symbolisch (Variablennamen), sondern abstrakt $\Sigma = \mathbf{Loc} \rightharpoonup \mathbf{Val}, \mathbf{Val} = \mathbf{N} + \mathbf{C} + \mathbf{Loc}$
  - Zustand wird als abstrakter Datentyp mit Operationen *Read* und *Upd* modelliert
  - Zuweisung nicht mehr mit Substitution/Ersetzung, sondern explizit durch *Upd*
  - Spezifikationen sind Zustandsprädikate
- Frame Conditions und Modification Sets
  - Frame Problem: welcher Teil des Zustands bleibt gleich?
  - Mit Zeigern: modification sets — Spezifikation des veränderlichen Teils

Ausblick

Die Sprache C: Was haben wir ausgelassen?

Semantik:

- Nichtdeterministische Semantik: Seiteneffekte, Sequence Points
  $\longrightarrow$ Umständlich zu modellieren, Effekt zweitrangig

- Implementationsabhängiges, unspezifiziertes und undefiniertes Verhalten
  $\longrightarrow$ Genauere Unterscheidung in der Semantik

Kontrollstrukturen:

- **switch** $\longrightarrow$ Ist im allgemeinen Fall ein **goto**

- **goto**, setjmp/longjmp
  $\longrightarrow$ Tiefe Änderung der Semantik (*continuations*)

Die Sprache C: Was haben wir ausgelassen?

Typen:

- Funktionszeiger $\longrightarrow$ Für "saubere" Benutzung gut zu modellieren
- Weitere Typen: **short**/**long int**, **double**/**float**, wchar_t, und Typkonversionen $\longrightarrow$ Fleißarbeit
- Fließkommazahlen $\longrightarrow$ Spezifikation nicht einfach
- **union** $\longrightarrow$ Kompliziert das Speichermodell
- **volatile** $\longrightarrow$ Bricht read/update-Gleichungen
- **typedef** $\longrightarrow$ Ärgernis für Lexer/Parser

Die Sprache C: Was haben wir ausgelassen?

Für realistische C-Programme:

- Compiler-Erweiterungen (`gcc`, `clang`)
- Büchereien (Standardbücherei, Posix, . . .)
- Nebenläufigkeit

Wie modelliert man Java?

- Die Kernsprache ist ähnlich zu C0.
- Java hat erschwerend
  - Dynamische Bindung,
  - Klassen mit gekapselten Zustand und Invarianten,
  - Nebenläufigkeit, und
  - Reflektion.
- Java hat dafür aber
  - ein einfacheres Speichermodell, und
  - eine wohldefinierte Ausführungsumgebung (die JVM).

Wie modelliert man PHP?

Gar nicht.

Korrekte Software in der Industrie

- Meist in speziellen Anwendungsgebieten: Luft-/Raumfahrt, Automotive, sicherheitskritische Systeme, Betriebssysteme
- Ansätze:
  1. Vollautomatisch: statische Analyse (Abstrakte Interpretation) für spezielle Aspekte: Freiheit von Ausnahmen und Unter/Überläufen, Programmsicherheit, Laufzeitverhalten (WCET)
     - Werkzeuge: `absint`
  2. Halbautomatisch: Korrektheitsannotationen, Überprüfung automatisch (nicht immer sound, aber vollständig)
     - Werkzeuge: JML (ESC/Java, Krakatao; Java), Boogie und Why (generisches VCG), Frama-C (C), VCC (C), Spark (ADA)
  3. Interaktiv: Einbettung der Sprache in interaktiven Theorembeweiser (Isabelle, Coq)
     - Beispiele: L4.verified, VeriSoft, SAMS

Feedback

Deine Meinung zählt

- Was war gut, was nicht?
- Arbeitsaufwand?
- Mehr Theorie oder mehr Praxis?
- Mehr oder weniger Isabelle?
- Mehr oder weniger Scala?

Tschüß!