Korrekte Software: Grundlagen und Methoden
Vorlesung 13 vom 03.07.18: Rückblick & Ausblick

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2018

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Die Floyd-Hoare-Logik
- Invarianten und die Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Modellierung und Spezifikation
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Funktionen und Prozeduren
- Referenzen
- 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
- Welche Semantik wofür?
- Beweis: Äquivalenz von operationaler und denotationaler Semantik

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)
- VCG: Schwächste Vorbedingung und stärkste Nachbedingung
- Beweis: Korrektheit und Vollständigkeit der Floyd-Hoare-Logik

Erweiterung der Programmiersprache

- Für jede Erweiterung:
  - Wie modellieren wir semantisch?
  - Wie ändern sich die Regeln der Logik?
- Strukturen und Felder
  - Lokationen: strukturierte Werte **Lexp**
  - Erweiterte Substitution in Zuweisungsregel

Erweiterung der Programmiersprache

- 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
- Referenzen
  - Konversion zwischen **Lexp**und **Aexp**
  - Lokationen nicht mehr symbolisch (Variablennamen), sondern abstrakt $\Sigma = \mathbf{Loc} \rightharpoonup \mathbf{V}, \mathbf{V} = \mathbb{Z} + \mathbf{Loc}$
  - Zustand als **abstrakter Datentyp** mit Operationen *read* und *upd*
  - Zuweisung nicht mehr mit Substitution, sondern explizit durch *upd*
  - Spezifikationen sind **explizite Zustandsprädikate**, Konversion $(-)^\dagger, (-)^\#$

Ausblick

Was geht noch?

- Die Sprache C
- Andere Programmiersprachen
- Logik und Spezifikation
- Success Stories

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
  $\longrightarrow$ Allgemeinfall: 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, sonst harmlos

Die Sprache C: Was haben wir ausgelassen?

Für **realistische C-Programme**:

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

Andere Sprachen: 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).

Andere Sprachen: Wie modelliert man C++?

- Sehr **vorsichtig** (konservativ)
- Viele Features, fehlende formale Semantik, ...
- Mehrfachvererbung theoretisch anspruchsvoll
- Es gibt **keine** Formalismen/Werkzeuge, die C++ voll unterstützen
- Ansätze: Übersetzung nach C/LLVM, Behandlung dort

Andere Sprachen: Wie modelliert man PHP?

Gar nicht.

Logik und Spezifikation

- Wir **generieren** Verifikationsbedingungen, wie kann man sie **beweisen**?
- **Automatische Beweiser:**
  - **SAT-Checker** lösen Erfüllbarkeitsproblem der Aussagenlogik (MiniSAT, Chaff)
  - **SMT-Beweiser** beweisen Aussagen der Prädikatenlogik mit linearer Arithmetik, Funktionen und Induktion (Z3, Yices, CVC)
- **Interaktive Beweiser:**
  - Beweisführung durch Benutzer, **Überprüfung** durch Beweiser
  - Sehr **mächtige** Logiken, aber nicht vollautomatisch (Isabelle, Coq)

Beispiel: Z3

- SMT-Beweiser versuchen Gegenbeweis zu konstruieren
- Daher: um ϕ zu beweisen, versuchen wir $\not\Phi$ zu widerlegen

Beweis einer VC:

$$x \geq 0 \wedge y > 0 \Longrightarrow x = 0 * y + x$$

Input Z3:

```
(declare-const x Int)
(declare-const y Int)
(assert
   (not (=> (and (>= x 0) (> y 0))
            (= x (+ (* 0 y) x))))
)
(check-sat)
```

Antwort:

`unsat`

Unerfüllbare VC:

$$x \geq 0 \wedge y > 0 \Longrightarrow x \geq y$$

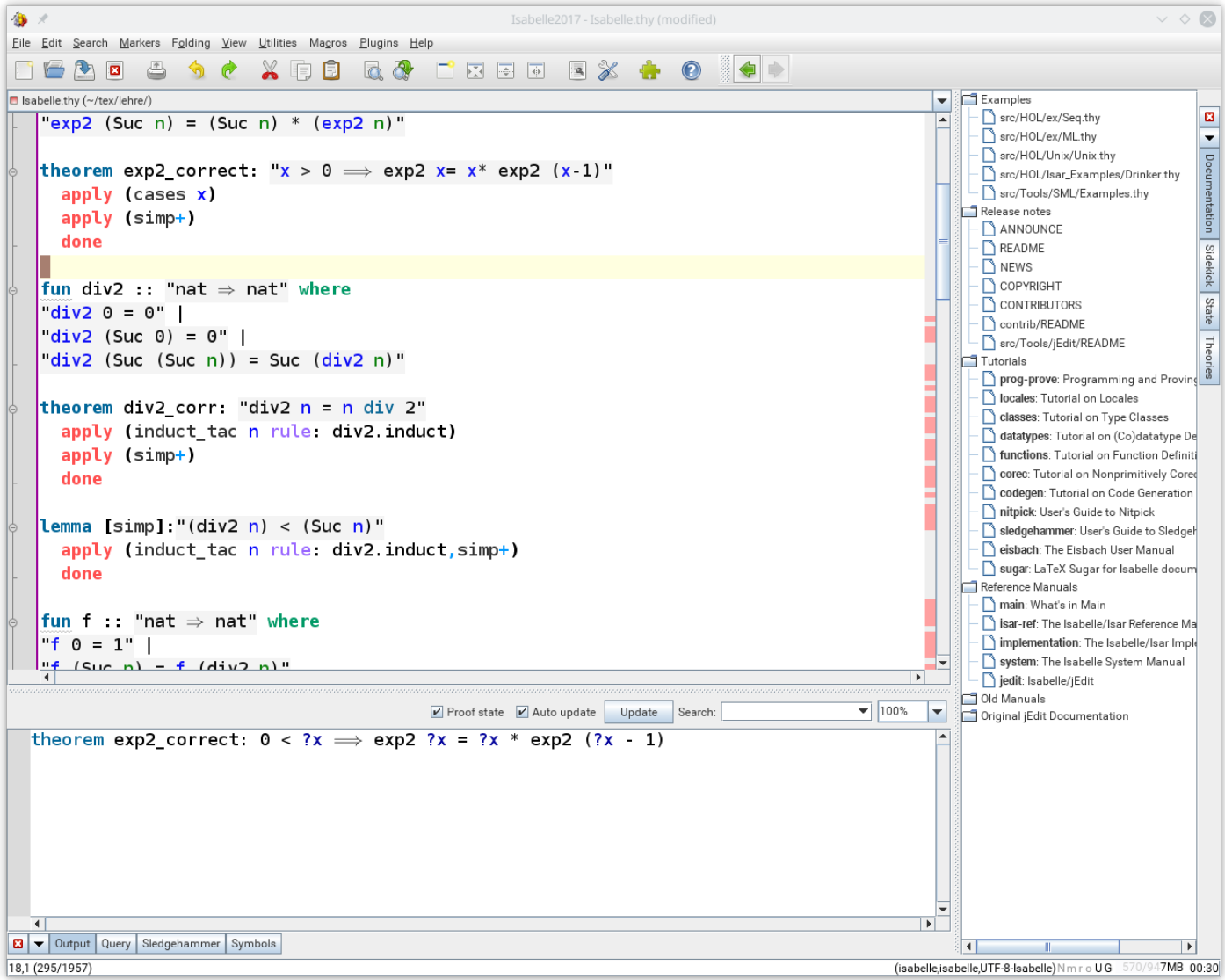
Input Z3:

```
(declare-const x Int)
(declare-const y Int)
(assert
   (not (=> (and (>= x 0) (> y 0))
            (>= x y)))
)
(check-sat)
```

Antwort:

`sat`

Beispiel: Isabelle



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) (nicht immer korrekt, meist vollständig)
     - Werkzeuge: `absint`
  2. Halbautomatisch: **Korrektheitsannotationen**, Überprüfung automatisch
     - Werkzeuge: Spark (ADA), Frama-C (C), JML (ESC/Java, Krakatao; Java), Boogie und Why (generisches VCG), VCC (C)
  3. Interaktiv: Einbettung der Sprache in interaktiven Theorembeweiser (Isabelle, Coq)
     - Beispiele: L4.verified, CompCert, SAMS

Feedback

Deine Meinung zählt

- Was war gut, was nicht?
- Arbeitsaufwand?
- Mehr **Theorie** oder mehr **Praxis**?
- Programmieraufgaben?
- Leichtgewichtiger Übungsbetrieb — mehr oder weniger?
- Bitte auch die **Evaluation** auf stud.ip beantworten!

Tschüß!