Technische Hochschule Rosenheim

Programmcodevalidierung mit Coq

Lukas Kiederle
Fakultät für Informatik

WS 2019/20

Kurzfassung

Aus Zeit- und Kostengründen beim Entwickeln und Testen von komplexen Systemen werden Tools zur Programmcodevalidierung immer relevanter. Diese Tools ermöglichen das Schreiben von Programmen, welche mathematisch und maschinell geprüft sind. Dadurch ist sichergestellt, dass das beschriebene Programm sich auch wie gewünscht, verhält.
Ziel dieser Arbeit ist es, einen sowohl theoretischen als auch technischen Einblick in die Programmcodevalidierung mit dem Proof Assistent Tool Coq darzustellen. Als Einstieg werden die grundlegende Begriffe geklärt und ein kurzer Überblick über Tools in diesem Fachbereich dargestellt. Dabei wird insbesondere auf Coq eingegangen.
Um ein Verständnis zu bekommen, wie ein Proof Assistent Tool die Qualität von Programmcode sicherstellt, müssen zunächst die Grundlagen dieser Sprache anhand von Beispielen erklärt werden. Anschließend wird näher auf das Zusammenspielen zwischen Programmcode und Proof Assistent eingegangen.
Es gibt bereits einige sehr erfolgreiche Forschungsprojekte, die Coq im Einsatz haben. Diese werden abschließend vorgestellt. Schlussendlich wird ein Fazit inklusive Ausblick in Hinsicht auf die Verwendbarkeit von Proof Assistent Tools gezogen.

Schlagworte:

- Proof Assistant
- Coq
- Programcodevalidierung

Leseanleitung

Hinweise auf referenzierte Literatur und die daraus entnommenen Zitate, welche in eckigen Klammern angegebenen sind, werden im Literaturverzeichnis am Ende der Arbeit aufgeführt. Soll ein Begriff oder eine Formulierung besonders hervorgehoben werden, ist diese *kursiv* geschrieben. Abkürzungen werden bei erstmaligem Auftreten einmal in runden Klammern, anschließend an das Wort ausgeschrieben. Um den Lesefluss nicht zu stören, werden alle darauf folgenden Wiederholungen der Abkürzungen nicht immer explizit ausgeschrieben.
Möglicherweise unbekannte Begriffe und Fachbegriffe werden bei ihrer ersten Nennung **fett** gedruckt. Diese sind im Glossar in alphabetischer Reihenfolge aufgelistet und werden näher erklärt. Einzige Ausnahme hierbei sind Überschriften von Tabellen. Um ein zusammenhängendes Lesen der Arbeit zu erleichtern, werden bei Bedarf Erklärungen bereits im Text gegeben. Dabei wird davon ausgegangen, dass der Leser bereits mit grundlegenden Begriffen der Informatik vertraut ist. Ausgehend vom Wissensstand eines entsprechend vorgebildeten Lesers, werden demzufolge nur fachlich speziellere Begriffe erklärt.
Um Unklarheiten zu vermeiden, werden Fachbegriffe in und zur Beschreibung von Bildern und Prozessen in ihrer originalen Sprache Englisch verwendet und nicht immer übersetzt.
An den Stellen, an denen es der Ausführung des Textes dient, sind kurze Codebeispiele im Text eingebunden. Außerdem werden Abbildungen zur Veranschaulichung verwendet, um komplexe Prozesse einfacher und verständlicher zu machen. Größere Abbildungen befinden sich im Anhang und werden im passenden Textabschnitt referenziert. Damit soll der Lesefluss nicht gestört werden.

Inhaltsverzeichnis

1 Motivation

Heutige Software wird üblicherweise so gebaut, dass sie funktioniert. Das bedeutet, dass eine Person, welche mit der Software interagiert, diese möglichst problemfrei nutzen kann. Dabei spielt oftmals vor alle in sicherheitskritischen Bereichen die Qualität eine große Rolle. Um diese zu gewährleisten, werden verschiedenste Methoden wie zum Beispiel Unit-Tests, Integrationstests und manuelles Testen eingesetzt.
Ein Problem bei dieser Art Software zu testen, ist allerdings, wenn Fälle außerhalb der bereits bekannten Tests aufkommen. Dies könnte für die jeweilige Software bedeuten, dass es zu keinem Problem bis hin zu großen finanziellem oder auch menschlichem Schaden kommen kann.
Mathematisch und maschinell geprüfte Programme sollen diese Lücke in Zukunft schließen.

- Normale Software wird so gebaut und evaluiert, sodass sie funktioniert
- "If you start with an English-language specification, you're inherently starting with an ambiguous specification," said Jeannette Wing, corporate vice president at Microsoft Research. "Any natural language is inherently ambiguous. In a formal specification you're writing down a precise specification based on mathematics to explain what it is you want the program to do."
- `https://www.quantamagazine.org/formal-verification-creates-hacker-proof-cod`
- Fehler 40
- Was wenn ein Compiler auf unterschiedlichen System aus dem selben Sourcecode unterschiedliche Runnables macht?
- `https://www.mikrocontroller.net/articles/Compilerfehler`
- `https://blog.regehr.org/archives/26`
- Unit tests schreiben, um möglichst viele Fehler zu finden -> Proof Assistent um für alle korrekt zu funktionieren
- Proof Assistent benutzen um formal richtigen Code zu schreiben/generieren
- **Fragestellung: Wie gewährleiste ich sichereren/gut getesteten Code?**

2 Grundlagen

2.1 Was ist ein Proof Assisstant

`https://www.youtube.com/watch?v=95VlaZTaWgc&t=2646s`

2.1.1 Proof Verifier

2.1.2 Theorem Provers

2.2 Übersicht

`https://en.wikipedia.org/wiki/Proof_assistant`

- ACL2
- Isabelle
- Coq

3 Coq

https://www.amazon.de/Certified-Programming-Dependent-Types-Introduction/dp/0262026651/ref=sr_1_fkmr0_1?__mk_de_DE=ÅMÅŽ~O~N&keywords=coq+proof+assistent&qid=1572974699&sr=8-1-fkmr0

- Warum Coq
- funktionaler Programmierstil
- Dependent type Sprache
- Frenchmade
- Warum ist coq so erfolgreich?

4 Programmatische Coq-Grundlagen

4.1 funktionaler Programmierstil

4.2 Basisbegriffe

https://softwarefoundations.cis.upenn.edu/lf-current/Basics.html#lab18

- Funktion
- Taktik
- Ziel
- Subgoal
- Proof/Beweis

4.3 Sprache

4.4 Beispielbeweise

```
1         Inductive day : Type :=
2         | monday
3         | tuesday
4         | wednesday
5         | thursday
6         | friday
7         | saturday
8         | sunday.
9
```

```
10          Definition next_weekday (d:day) : day :=
11          match d with
12          | monday => tuesday
13          | tuesday => wednesday
14          | wednesday => thursday
15          | thursday => friday
16          | friday => monday
17          | saturday => monday
18          | sunday => monday
19          end.
20
21          Compute (next_weekday friday).
22          (* ==> monday : day *)
23          Compute (next_weekday (next_weekday saturday)).
24          (* ==> tuesday : day *)
```

Codebeispiel 1: Coq Beispiel

5 Zusammenspiel Proof - Program

- Wir haben eine Liste von Dingen, die die Software tun soll, und verwenden Logik, um zu beweisen, dass die Software diese Dinge tut.
- `https://www.youtube.com/watch?v=Ue8QG8pf0wU`
- Codegenerierung, Proof -> Program
- Programm -> Proof
- Beispiel mit Liste aus Video

5.1 How-to

- `https://medium.com/@ahelwer/formal-verification-casually-explained-3fb4fef2`
- Erstelle eine Spezifikation, die ein Programm beschreibt
- Schreie die Spezifikation mathematisch in ein Proof Tool
- Wenn es Fehler enthält, sagt es dir der Verifier
-

6 Aktuelle Anwendung

6.1 Proofed Stack

- CompCert (C compiler)
- Princeton VST

- Certikos (verified Operating System with hypervisor and multi instances)
- http://plv.csail.mit.edu/kami/
- https://www.zdnet.com/article/certikos-a-hacker-proof-os/
- https://github.com/PrincetonUniversity/VST
- https://vst.cs.princeton.edu
- https://news.yale.edu/2016/11/14/certikos-breakthrough-toward-hacker-resist

6.2 CompCert for C

http://compcert.inria.fr

6.3 JSCert for ECMA 5

https://github.com/jscert/jscert

6.4 4-Farben Rätsel ist löstbar!

6.5 CertiCoq

https://www.cs.princeton.edu/~appel/papers/certicoq-coqpl.pdf

7 Anwendbarkeit in der Praxis

- Objektorientierung
- Sicherheitskritische Systeme
- Compilerbau

8 Fazit

8.1 Aussicht

9 Glossar

10 Einleitung

Hier kommt die Einleitung.

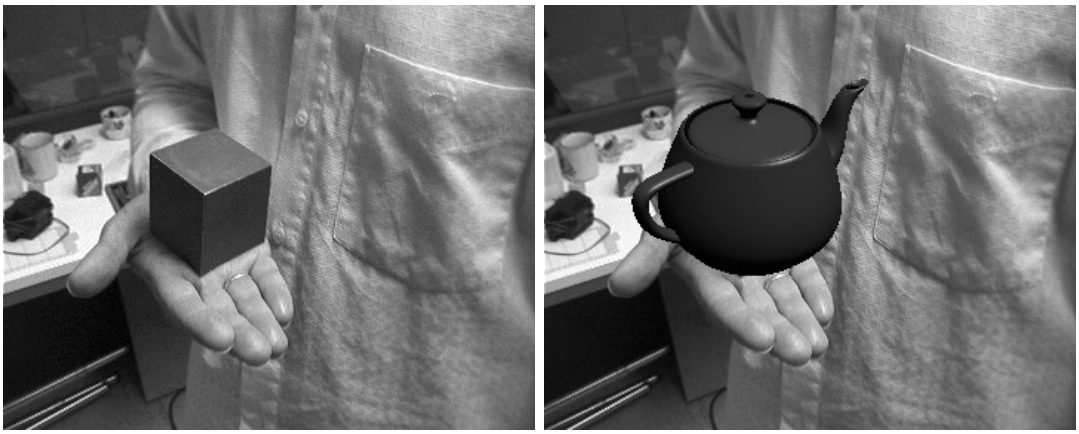
10.1 Ein Abschnitt der Einleitung

Einen Überblick findet man z. B. in [Aue00].

Ein Beispiel wird in Abb. 1 gezeigt. Das verwendete Objekt ist in Abb. 1a dargestellt, das Ergebnis in Abb. 1b.

Eine Formel

$$f(x) = \frac{1}{3}x + 5, \quad x \in \mathbb{R}. \tag{1}$$



(a) Originalbild (b) erweitertes Bild

Abbildung 1: Beispiel eines Augmented Reality Systems: es folgt eine Beschreibung (Bilder aus [Sch01])

Sequence	ARTS	wman	stcams	ARTVZ	ARTSUZ
# Frames	190	40	400	270	190
# relative movements	17955	780	79800	36315	17955
# movements after pre-sel.	14336	623	37915	21788	14343
min. angle in seq.	0.233°	5.95°	0.154°	0.00000171°	0.0388°
max. angle in seq.	81.7°	180°	47.3°	80.3°	80.9°
min. angle after pre-sel.	12.9°	21.1°	17.3°	16.3°	12.9°
max. angle after pre-sel.	81.7°	161°	47.3°	80.3°	80.9°

Tabelle 1: Datenselektion für verschiedene Testdatensätze.

Und noch eine:

$$\boldsymbol{M} = \boldsymbol{A}\boldsymbol{x}\pi, \quad \boldsymbol{A} \in \mathbb{R}^{2\times 2}, \boldsymbol{x} \in \mathbb{R}^2. \tag{2}$$

Tabelle 1 gibt einen Überblick über XYZ.

A Erster Abschnitt des Anhangs

In diesem Anhang wird ...

Literatur

[Aue00] T. Auer. *Hybrid Tracking for Augmented Reality.* Dissertation, Technische Universität Graz, Graz, Austria, 2000.

[Sch01] J. Schmidt, I. Scholz und H. Niemann. Placing Arbitrary Objects in a Real Scene Using a Color Cube for Pose Estimation. In B. Radig und S. Florczyk, Hg., *Pattern Recognition, 23rd DAGM Symposium*, Bd. 2191 von *Lecture Notes in Computer Science*, S. 421–428. Springer-Verlag, Berlin, Heidelberg, New York, 2001.