

Praktikum Spezifikation und Verifikation

1 Hoare Logik

Die Beschreibung imperativer Programme durch sogenannte Hoare-Tripel eignet sich gut, um Eigenschaften von Programmen zu beweisen. Hierbei werden Programmkonstrukte mit Zusicherungen (= Prädikate über die Programmvariablen) annotiert, die das Verhalten des Programms beschreiben. $\{P\} c \{Q\}$ ist genau dann ein gültiges Hoare-Tripel, wenn für das Programm c und die Vorbedingung P nach Ausführung von c die Nachbedingung Q gilt (siehe z.B.[1]).

Für Isabelle/HOL existiert eine Definition der Hoare-Logik für eine einfache imperative Sprache mit folgenden syntaktischen Konstrukten:

$$\begin{aligned} c &= \text{SKIP} \\ &| x := a \\ &| c_0; c_1 \\ &| \text{IF } b \text{ THEN } c_0 \text{ ELSE } c_1 \text{ FI} \\ &| \text{WHILE } b \text{ INV } \{I\} \text{ DO } c \text{ OD} \\ p &= \{P\} c \{Q\} \end{aligned}$$

Wie man sieht, genügt es, innerhalb eines Programms nur den Rumpf von *WHILE*-Schleifen mit einer Schleifeninvariante I zu annotieren, für die anderen Konstrukte lassen sich die schwächsten Vorbedingungen aus den jeweiligen Nachbedingung automatisch ableiten. Die Taktik *hoare_tac* erzeugt für ein Hoare-Tripel $\{P\} c \{Q\}$ Verifikationsbedingungen, durch die sich die Gültigkeit des Tripels beweisen läßt.

Beispiele finden Sie unter <http://www.in.tum.de/~isabelle/library/HOL/Hoare>

Wichtige Informationen finden Sie unter

<http://www4.in.tum.de/~isabelle/library/HOL/Hoare/README.html>

Im Folgenden sollen einige imperative Programme mit Hilfe der Hoare Logik verifiziert werden. Die zugrundeliegende Theorie ist in [4] genauer beschrieben. Weitere Informationen zur axiomatischen Semantik finden sich z.B. in [1, §5.7.1] [5, §6] oder [2, §4].

Hier ein einfaches Beispiel zur Anwendung der Hoare Logik. Wir verifizieren ein Programm, das zwei natürliche Zahlen durch wiederholte Addition multipliziert.

Zuerst deklarieren wir die verwendeten Programmvariablen als Record-Typ.

```
record mult_vars =  
  M :: nat  
  N :: nat
```

Die zu beweisende Aussage wird als Hoare-Tripel wie folgt formuliert. Man beachte die Unterscheidung von Programmvariablen von einfachen Werten der Logik (z.B. \dot{M} vs. a).

```
theorem  
"/- .{ $\dot{M} = 0 \wedge \dot{N} = 0$ }.  
  WHILE  $\dot{M} \dot{=} a$   
  INV .{ $\dot{N} = \dot{M} * b$ }.  
  DO  $\dot{N} := \dot{N} + b; \dot{M} := \dot{M} + 1$  OD  
  .{ $\dot{N} = a * b$ }."
```

Die Hauptarbeit des Beweises erledigt die *hoare* Methode, welche ein mit Invarianten annotiertes Hoare-Tripel auf ein rein logisches Problem reduziert. Den Rest kann man in diesem Beispiel einfach mit *auto* erledigen.

```
apply hoare  
apply auto  
done
```

1.1 Quotient und Rest

Geben Sie ein annotiertes imperatives Program zur Berechnung des Quotients und Rests zweier natürlichen Zahlen x und y . Beweisen Sie dann, daß folgendes Hoare-Tripel gültig ist:

```
record quo_vars =  
  quo :: nat  
  rest :: nat
```

```
theorem "/- .{True}. program .{ $\dot{quo} * y + \dot{rest} = x \wedge \dot{rest} < y$  }."  
<proof>
```

1.2 Minimumsuche

Gesucht ist ein verifiziertes imperatives Programm das den Index des Minimums einer nicht-leeren ganzzahligen Liste A berechnet. Der gefundene Index soll in einer Variable j gespeichert werden.

Die Notation $A!j$ bezeichnet den Zugriff auf die j -te Komponente einer Liste A .

```
record list_vars =  
  j :: nat
```

```
theorem  
  "|- .{0 < length A}.  
    program  
    .{ $\forall i < \text{length } A. A!i \leq A!i$ }. "  
  <proof>
```

Hinweis: Man beachte, daß für die Sprache der Zusicherungen die volle Ausdrucksstärke von HOL zur Verfügung steht.

Hinweis: Wird die Einführung von Hilfsvariablen im Programm benötigt, müssen diese auch im Record der Programm-Variablen deklariert werden.

1.3 Die Fibonacci Funktion

Schreiben Sie ein WHILE-Programm, das die n -te Fibonacci Zahl iterativ berechnet.

Die Spezifikation verwendet dabei eine rekursive Definition der Fibonacci Funktion fib , die Sie in [3] finden können.

Ein imperatives Programm zur Berechnung der n -ten Fibonacci Zahl sei wie folgt spezifiziert:

```
record Fib_vars =  
  Fib :: nat
```

```
theorem "|- .{0 ≤ n}. program .{`Fib = fib(n)}. "  
  <proof>
```

Hinweis: Um mit der Multiplikation besser zurecht zu kommen, bieten sich u.a. folgende Simplifikationsregeln an:

```
thm add_mult_distrib  
thm add_mult_distrib2
```

Simplifikations-Regeln werden von links nach rechts angewendet. Falls Sie die umgekehrte Richtung einer Gleichung verwenden möchten, können Sie diese durch Verknüpfung mit dem Theorem sym z. B. folgendermaßen erhalten:

```
thm add_mult_distrib [THEN sym]
```

Setzen Sie für *program* einen geeigneten Text ein und beweisen Sie die Behauptung mit Hilfe der Hoare-Logik. Dabei dürfen im Programm nur die arithmetische Operationen $+$ und $-$ vorkommen.

Rekursive Version

Definieren Sie einen rekursiven Algorithmus für die Fibonacci Funktion, die wie die iterative Version auch nur lineare Komplexität hat. Beweisen Sie die Gültigkeit vom selben Hoare-Triple.

Literatur

- [1] M. Broy. *Informatik — Eine grundlegende Einführung (Teil I)*. Springer, 1992.
- [2] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science — Modelling and reasoning about systems*. Cambridge University Press, 2000. <http://www.cs.bham.ac.uk/research/lics/>.
- [3] T. Nipkow and L. Paulson. *Isabelle/HOL. The Tutorial*. 2001. Available at <http://isabelle.in.tum.de/doc/tutorial.pdf>.
- [4] M. Wenzel. A formulation of Hoare Logic in Isabelle/Isar, June 2000. <http://www4.in.tum.de/~wenzelm/papers/Hoare-Isar.pdf>.
- [5] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

▷ Abgabe Quotient und Rest, Minimum: 20. Juni 2001

▷ Abgabe Fibonacci: 27. Juni 2001