

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* erzeugt für ein Hoare-Tripel $\{P\} c \{Q\}$ Verifikationsbedingungen, durch die sich die Gültigkeit des Tripels beweisen läßt.

Beispiele finden Sie unter <http://www4.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 [3] genauer beschrieben. Weitere Informationen zur axiomatischen Semantik finden sich z.B. in [1, §5.7.1] [4, §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. \acute{M} vs. a).

theorem

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

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
```

Arrays

In dieser Übung werden wir Hoare-Logik verwenden, um die partielle Korrektheit imperativer Programme, die auf Arrays arbeiten, zu beweisen. Arrays werden in Isabelle als Listen modelliert. Der Operator $!$ kann verwendet werden, um einzelne Elemente einer Liste über ihren Index zu selektieren, z.B.:

```
lemma "[a,b,c,d] ! 2 = c"  
by simp
```

Der folgende Algorithmus implementiert serielle Suche in einem Array:

```
types val = nat  
record vars =  
  i :: nat  
  v :: val  
  A :: "val list"
```

```

lemma SerialSearch:" $\vdash$ 
  {length `A = n  $\wedge$  `A = I}
  `i := 0;
  WHILE `i < n  $\wedge$  `A ! `i  $\neq$  `v
    INV {`A = I  $\wedge$  length `A = n}
    DO
      `i := `i + 1
    OD
  {length `A = n  $\wedge$  `A = I  $\wedge$  (n  $\leq$  `i  $\vee$  `A ! `i = `v)}"
  :
```

- Beweisen Sie die Unterziele, die bei diesem Hoare-Tripel auftreten.
- Fügen Sie der Nachbedingung die Aussage ($\text{`v} \in \text{set `A} \longrightarrow \text{`A ! `i} = \text{`v}$) hinzu. Beweisen Sie die Unterziele dieses neuen Hoare-Tripels.
Hinweis: Sie werden auch die Schleifeninvariante ändern müssen.
- Beweisen Sie jetzt das entsprechende Hoare-Tripel für serielle Suche in einer sortierten Liste:
 - Nehmen Sie an, daß das gegebene Array sortiert ist.
 - Ändern Sie die Vorbedingung entsprechend.
 - Ändern Sie die Schleifenbedingung, um durch Ausnützen der Tatsache, daß die Liste sortiert ist, möglicherweise die Laufzeit des Algorithmus zu verkürzen.
 - Beweisen Sie ($\text{`v} \in \text{set `A} \longrightarrow \text{`A ! `i} = \text{`v}$) als eine Nachbedingung.

Hinweise:

- Definieren Sie eine Funktion *sorted*: $\text{val list} \Rightarrow \text{bool}$, die überprüft, ob eine Liste sortiert ist.
- Machen Sie sich mit den Funktionen *take* und *drop* in der Isabelle Listen-Bibliothek vertraut.

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] M. Wenzel. A formulation of Hoare Logic in Isabelle/Isar, June 2000. <http://www4.in.tum.de/~wenzelm/papers/Hoare-Isar.pdf>.
- [4] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

▷ **Abgabe: 23. Mai 2003**