

Praktikum Spezifikation und Verifikation

Natürliches Schließen

In dieser Aufgabe geht es um den Kalkül des natürlichen Schließens, mit dessen Hilfe einige Lemmas der Aussagen-Logik bewiesen werden sollen.

Für die Beweise gelten die folgenden Spielregeln:

- Es dürfen nur diese Lemmas verwendet werden:
 $notI: (A \implies False) \implies \neg A,$
 $notE: [\neg A; A] \implies B,$
 $conjI: [A; B] \implies A \wedge B,$
 $conjE: [A \wedge B; [A; B] \implies C] \implies C,$
 $disjI1: A \implies A \vee B,$
 $disjI2: A \implies B \vee A,$
 $disjE: [A \vee B; A \implies C; B \implies C] \implies C,$
 $impI: (A \implies B) \implies A \longrightarrow B,$
 $impE: [A \longrightarrow B; A; B \implies C] \implies C,$
 $mp: [A \longrightarrow B; A] \implies B$
 $iffI: [A \implies B; B \implies A] \implies A = B,$
 $iffE: [A = B; [A \longrightarrow B; B \longrightarrow A] \implies C] \implies C$
 $classical: (\neg A \implies A) \implies A$

- Es dürfen nur die Methoden *rule*, *erule* und *assumption* verwendet werden.

lemma I: " $A \longrightarrow A$ "

⋮

lemma " $A \wedge B \longrightarrow B \wedge A$ "

⋮

lemma " $(A \wedge B) \longrightarrow (A \vee B)$ "

⋮

lemma K: " $A \longrightarrow B \longrightarrow A$ "

⋮

```

lemma "(A ∨ A) = (A ∧ A)"
:
lemma S: "(A → B → C) → (A → B) → A → C"
:
lemma "(A → B) → (B → C) → A → C"
:
lemma "¬ ¬ A → A"
:
lemma "(¬ A → B) → (¬ B → A)"
:
lemma "((A → B) → A) → A"
:
lemma "A ∨ ¬ A"
:

```

Hoare-Logik

Die Beschreibung imperativer Programme durch 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:

```

c = SKIP
  | x := a
  | c0; c1
  | IF b THEN c0 ELSE c1 FI
  | WHILE b INV {I} DO c OD
= {P} c {Q}

```

Wie man sieht, genügt es, innerhalb eines Programms 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 Methode *hoare* 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 [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: Programmvariablen wie \acute{M} hängen implizit vom aktuellen Zustand s des Programms ab, logische Variablen wie a nicht.

theorem

```
"⊢ .{ $\acute{M} = 0 \wedge \acute{N} = 0$ }.  
  WHILE  $\acute{M} \neq a$   
  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
```

Quotient und Rest

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

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

```
theorem "⊢ .{True}. programm .{ $\acute{quo} * y + \acute{rest} = x \wedge \acute{rest} < y$  }."  
⋮
```

Sortieren

Implementieren Sie die Funktion *insort* des Sortieralgorithmus aus Aufgabenblatt 2 imperativ. Eingabe ist eine Liste *ls*, Ausgabe eine neue Liste *ss*, in die das Element *n* geordnet eingefügt wurde. Zeigen Sie folgendes Hoare-Tripel:

```
record insort_vars =  
  ls :: "nat list"  
  ss :: "nat list"
```

```
theorem "⊢ .{`ls = list}. programm .{`ss = insort n list}."  
  :
```

▷ Hinweise:

- Die Eingabeliste *ls* darf verändert werden.
- Um eine verkettete Liste zu simulieren, dürfen Sie im Programm selbst nur die Funktionen *hd*, *tl*, und *@ [x]* (append mit einer einelementigen Liste) benutzen. In den Invarianten steht ihnen die volle Mächtigkeit von HOL mit allen Funktionen und Ausdrücken zur Verfügung.
- Sie werden für die Invariante evtl. eine neue Funktion definieren müssen.

Freiwillige Zusatzaufgabe:

Implementieren Sie den gesamten Sortieralgorithmus (die Funktion *sort* von Aufgabenblatt 2) imperativ, und beweisen Sie, dass er das gleiche Ergebnis liefert wie *sort*. Sie können alternativ auch zeigen, dass für das Ergebnis *sorted* gilt, und *count* nicht verändert wird.

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 Natürliches Schließen: 23. Mai 2002

▷ Abgabe Quotient + Sortieren: 29. Mai 2002