Beweisstile im Hoare-Kalkül

Moritz Hammer
(hammer@pst.ifi.lmu.de)

18. Mai 2006

Wir betrachten folgendes Programm, um Primzahlen zu prüfen (eigentlich wird die Teilbarkeit von `c` geprüft und in `d` gespeichert):

$\{\mathtt{c} > 1\}$

```
boolean d = false;
int i = 2;

while (i < c) {
    d = d | (c % i == 0);
    i = i + 1;
}
```

$\{\mathtt{d} \Leftrightarrow \exists\ 2 \leq j < \mathtt{c}\ .\ \mathtt{c} \bmod j = 0\}$

Das Vorgehen beim Beweis ist folgendermassen:

Wir beginnen mit der Invariante. Da die `while`-Schleife wieder nur einen Ersatz für eine `for`-Schleife darstellt, bietet sich an, zu beschreiben, wie die Zählervariable `i` Einfluss auf den „Zustand“ der Schleife hat. Wir betrachten also den Effekt der Schleife: `d` ist genau dann gesetzt, wenn `d` vorher schon gesetzt war, oder wenn `c` durch `i` teilbar ist. Durch ein Induktionsargument können wir uns überlegen, dass `d` genau dann gesetzt ist, wenn `c` durch eine Zahl zwischen 1 und `i` teilbar ist. Insofern gilt vor wie nach Ausführung der Schleife, dass alle Zahlen $2 \leq j < \mathtt{i}$ geprüft wurden. Ein Teil der Invariante ist demnach $\mathtt{d} \Leftrightarrow (\exists\ 2 \leq j < \mathtt{i}\ .\ \mathtt{c} \bmod j = 0)$.

Dies reicht aber noch nicht: Wir müssen sicherstellen, dass die Nachbedingung $\{\mathtt{d} \Leftrightarrow \exists\ 2 \leq j < \mathtt{c}\ .\ \mathtt{c} \bmod j = 0\}$ aus der Invariante folgt. Nach der (Iteration$_{\text{partiell}}$)-Regel erhalten wir aber als Nachbedingung der `while`-Schleife jedoch nur $\{I \wedge \neg(\mathtt{i} < \mathtt{c})\}$, also $I \wedge \mathtt{i} > \mathtt{c}$. Wir müssen also, um am Ende $\mathtt{i} = \mathtt{c}$ folgern zu können, noch $\mathtt{i} \leq \mathtt{c}$ in die Invariante aufnehmen, denn $\mathtt{i} \leq \mathtt{c} \wedge \mathtt{i} \geq \mathtt{c} \implies \mathtt{i} = \mathtt{c}$.

Zuletzt müssen wir uns noch gegen ein Problem absichern: Wie wir später sehen werden, müssen wir mit dem Term $(\mathtt{c} \bmod \mathtt{i})$ rechnen. Dieser ist aber undefiniert für $\mathtt{i} = 0$. Insofern müssen wir noch $i > 0$ in die Invariante aufnehmen. Insgesamt erhalten wir

$$I \equiv (\mathtt{d} \Leftrightarrow (\exists\ 2 \leq j < \mathtt{i}\ .\ \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} \leq \mathtt{c} \wedge \mathtt{i} > 0$$

Fehlen Teile der Invariante, sollte es unmöglich sein, das Hoare-Tripel $\{I \wedge (\mathtt{i} < \mathtt{c})\}\ \ldots\ \{I\}$ herzuleiten. In diesem Fall muss man nachträglich die Invariante verstärken.

Der Beweis geht nun von unten nach oben:

Zuerst zeigen wir, dass die Invariante in der Tat invariant ist, also dass wir die (Iteration$_{\text{partiell}}$)-Regel anwenden dürfen. Dazu muß die Prämisse $\{I \wedge \mathtt{i} < \mathtt{c}\}\ \ldots\ \{I\}$ gezeigt werden. Dies machen wir, ausgehend von der Nachbedingung $\{I\}$ von unten nach oben:

$$\{(\mathtt{d} \Leftrightarrow (\exists\ 2 \leq j < \mathtt{i}\ .\ \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} \leq \mathtt{c} \wedge \mathtt{i} > 0\}$$
$$\texttt{i=i+1;}$$
$$\{(\mathtt{d} \Leftrightarrow (\exists\ 2 \leq j < \mathtt{i} + 1\ .\ \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} + 1 \leq \mathtt{c} \wedge \mathtt{i} + 1 > 0\}$$
$$\texttt{d = d | (c \% i == 0);}$$
$$\{(\mathtt{d} \vee (\mathtt{c} \bmod \mathtt{i} = 0) \Leftrightarrow (\exists\ 2 \leq j < \mathtt{i} + 1\ .\ \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} + 1 \leq \mathtt{c} \wedge \mathtt{i} + 1 > 0\}$$

Nun schreiben wir die letzte Zeile um, indem wir verwenden, dass $(\exists\ 2 \leq j < \mathtt{i} + 1\ .\ \varphi) \Leftrightarrow ((\exists\ 2 \leq j < \mathtt{i}\ .\ \varphi) \vee \varphi[j/\mathtt{i}])$ gilt (wir ziehen quasi das letzte Element, das beim $\exists$-Quantor geprüft wird, heraus):

$$\{(\mathtt{d} \vee (\mathtt{c} \bmod \mathtt{i} = 0) \Leftrightarrow ((\exists\ 2 \leq j < \mathtt{i}\ .\ \mathtt{c} \bmod j = 0) \vee \mathtt{c} \bmod \mathtt{i} = 0)) \wedge \mathtt{i} + 1 \leq \mathtt{c} \wedge \mathtt{i} + 1 > 0\}$$

Und dies wird von der Invariante und $\texttt{i} < \texttt{c}$ impliziert: Die Invariante enthält, dass `d` genau dann gesetzt ist wenn $\exists\ 2 \leq j < \texttt{i}.\texttt{c} \bmod j = 0$ gilt. Gilt also die Invariante, können wir `d` durch diese Formel ersetzen, und erhalten somit

$$\{(((\exists\ 2 \leq j < \texttt{i}\ .\ \texttt{c} \bmod j = 0) \vee \texttt{c} \bmod \texttt{i} = 0) \Leftrightarrow ((\exists\ 2 \leq j < \texttt{i}\ .\ \texttt{c} \bmod j = 0) \vee \texttt{c} \bmod \texttt{i} = 0))) \wedge \texttt{i} + 1 \leq \texttt{c} \wedge \texttt{i} + 1 > 0\}$$

Gilt also die Invariante, dann steht auf beiden Seiten der Äquivalenz das Gleiche (und damit ist sie wahr). Aus $\texttt{i} > 0$ folgt natürlich auch $\texttt{i} + 1 > 0$. Und $\texttt{i} < \texttt{c}$ ist äquivalent zu $\texttt{i} + 1 \leq \texttt{c}$. Gelten also die Invariante und $\texttt{i} < \texttt{c}$, dann gilt auch die letzte Zeile. Daher impliziert $I \wedge \texttt{i} < \texttt{c}$ die letzte Zeile, und wir können die (Iteration$_{\text{partiell}}$)-Regel anwenden.

Der Initialisierungsteil vor der `while`-Schleife macht keine Probleme mehr:

$$\begin{array}{c}
\{(\texttt{d} \Leftrightarrow (\exists\ 2 \leq j < \texttt{i}\ .\ \texttt{c} \bmod j = 0)) \wedge \texttt{i} \leq \texttt{c} \wedge \texttt{i} > 0\} \\
\texttt{int i = 2;} \\
\{(\texttt{d} \Leftrightarrow (\exists\ 2 \leq j < 2\ .\ \texttt{c} \bmod j = 0)) \wedge 2 \leq \texttt{c} \wedge 2 > 0\} \\
\texttt{boolean d = false;} \\
\{(\textbf{false} \Leftrightarrow (\exists\ 2 \leq j < 2\ .\ \texttt{c} \bmod j = 0)) \wedge 2 \leq \texttt{c} \wedge 2 > 0\}
\end{array}$$

Dies vereinfacht sich zu $2 \leq \texttt{c}$ (denn es gibt keine ganze Zahl, die echt zwischen 1 und 2 liegt, und daher gilt $\textbf{false} \Leftrightarrow \exists\ 2 \leq j < 2\ .\ \varphi$ für jede Formel φ, ausserdem gilt natürlich immer $2 > 0$) und dies ist äquivalent zu $\texttt{c} > 1$.

Der formelle Beweis

Im formellen Beweis arbeiten wir exakt mit dem Hoare-Kalkül. Dabei wird schnell der Platz ein Problem; insofern verwenden wir Abkürzungen und rechnen die Invariante nicht immer aus:

$$\mathbf{F_1} \equiv \left\{ \dfrac{\dfrac{}{\{(\texttt{d} \vee (\texttt{c} \bmod \texttt{i} = 0) \Leftrightarrow (\exists\ 2 \leq j < \texttt{i} + 1\ .\ \texttt{c} \bmod j = 0)) \wedge \texttt{i} + 1 \leq \texttt{c} \wedge \texttt{i} + 1 > 0\}\ \texttt{d = d | (c \% i == 0);}\ \{I[\texttt{i}/\texttt{i}+1]\}}\ (\text{Zuw})}{\{I \wedge \texttt{i} < \texttt{c}\}\ \texttt{d = d | (c \% i == 0);}\ \{I[\texttt{i}/\texttt{i}+1]\}}\ (\text{Abschw}) \right.$$

$$\mathbf{F}_{\text{while}} \equiv \left\{ \dfrac{\dfrac{\dfrac{\mathbf{F_1} \quad \dfrac{}{\{I[\texttt{i}/\texttt{i}+1]\}\ \texttt{i=i+1;}\ \{I\}}\ (\text{Zuw})}{\{I \wedge \texttt{i} < \texttt{c}\}\ \texttt{d = d | (c \% i == 0); i = i + 1;}\ \{I\}}}{\{I \wedge \texttt{i} < \texttt{c}\}\ \texttt{\{d = d | (c \% i == 0); i = i + 1;\}}\ \{I\}}\ (\text{Block})}{\{I\}\ \texttt{while(i < c) \{d = d | (c \% i == 0); i = i + 1;\}}\ \{I \wedge \neg(\texttt{i} < \texttt{c})\}}\ (\text{Iteration}_{\text{partiell}}) \right.$$

$$\mathbf{F}_{\text{Kopf}} \equiv \left\{ \dfrac{\dfrac{\dfrac{}{\{I[\texttt{i}/2][\texttt{d}/\textbf{false}]\}\ \texttt{boolean d=false;}\ \{I[\texttt{i}/2]\}}\ (\text{Zuw})}{\{c > 1\}\ \texttt{boolean d=false;}\ \{I[\texttt{i}/2]\}}\ (\text{Abschw}) \quad \dfrac{}{\{I[\texttt{i}/2]\}\ \texttt{int i=2;}\ \{I\}}\ (\text{Zuw})}{\{c > 1\}\ \texttt{boolean d=false; int i=2;}\ \{I\}}\ (\text{Seq}) \right.$$

$$\mathbf{F}_{\text{prim}} \equiv \left\{ \dfrac{\dfrac{\mathbf{F}_{\text{Kopf}} \quad \dfrac{\mathbf{F}_{\text{while}}}{\{I\}}\ (\text{Abschw})}{\texttt{while(i < c) \{d = d | (c \% i == 0); i = i + 1;\}}\ \{\texttt{d} \Leftrightarrow (\exists\ 2 \leq j < \texttt{c}\ .\ \texttt{c} \bmod j = 0)\}}\ (\text{Seq})}{\{c > 1\}\ \texttt{boolean d=false; int i=2;}\ \texttt{while(i < c) \{d = d | (c \% i == 0); i = i + 1;\}}\ \{\texttt{d} \Leftrightarrow (\exists\ 2 \leq j < \texttt{c}\ .\ \texttt{c} \bmod j = 0)\}} \right.$$

Allein die Platzerfordernisse machen diesen Modus unattraktiv. Er dient mehr der theoretischen Fundierung.

Beweisskizzen

Beweisskizzen folgen der Struktur des Codes. Man schreibt den Beweis, den man von unten nach oben durchrechnet, von oben nach unten hin. In unserem Fall ergibt sich hierbei:

$$
\begin{array}{l}
\{\mathtt{c} > 1\} \\
\quad \Downarrow \\
\{(\mathbf{false} \Leftrightarrow (\exists\, 2 \le j < 2 \,.\, \mathtt{c} \bmod j = 0)) \wedge 2 \le \mathtt{c} \wedge 2 > 0\} \\
\texttt{ boolean d = false;} \\
\{(\mathtt{d} \Leftrightarrow (\exists\, 2 \le j < 2 \,.\, \mathtt{c} \bmod j = 0)) \wedge 2 \le \mathtt{c} \wedge 2 > 0\} \\
\texttt{ int i = 2;} \\
\{(\mathtt{d} \Leftrightarrow (\exists\, 2 \le j < \mathtt{i} \,.\, \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} \le \mathtt{c} \wedge \mathtt{i} > 0\} \\
\texttt{ while (i < c) \{} \\
\{(\mathtt{d} \Leftrightarrow (\exists\, 2 \le < \mathtt{i} \,.\, \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} \le \mathtt{c} \wedge \mathtt{i} > 0 \wedge \mathtt{i} < \mathtt{c}\} \\
\quad \Downarrow \\
\{((\mathtt{d} \vee (\mathtt{c} \bmod \mathtt{i} = 0)) \Leftrightarrow (\exists\, 2 \le j < \mathtt{i}+1 \,.\, \mathtt{c} \bmod j = 0)) \wedge \mathtt{i}+1 \le \mathtt{c} \wedge \mathtt{i}+1 > 0\} \\
\qquad \texttt{d = d | (c \% i == 0);} \\
\{(\mathtt{d} \Leftrightarrow (\exists\, 2 \le j < \mathtt{i}+1 \,.\, \mathtt{c} \bmod j = 0)) \wedge \mathtt{i}+1 \le \mathtt{c} \wedge \mathtt{i}+1 > 0\} \\
\qquad \texttt{i = i + 1;} \\
\{(\mathtt{d} \Leftrightarrow (\exists\, 2 \le j < \mathtt{i} \,.\, \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} \le \mathtt{c} \wedge \mathtt{i} > 0\} \\
\texttt{ \}} \\
\{(\mathtt{d} \Leftrightarrow (\exists\, 2 \le j < \mathtt{i} \,.\, \mathtt{c} \bmod j = 0)) \wedge \mathtt{i} \le \mathtt{c} \wedge \mathtt{i} > 0 \wedge \neg(\mathtt{i} < \mathtt{c})\} \\
\quad \Downarrow \\
\{\mathtt{d} \Leftrightarrow (\exists\, 2 \le j < \mathtt{c} \,.\, \mathtt{c} \bmod j = 0)\}
\end{array}
$$

Anhand dieser Darstellung lässt sich die Korrektheit des Beweises leicht nachvollziehen.

Annotierte Programme

Das Problem hier ist, dass ohne weiteres keine `assert`-Zeilen generiert werden können: Für den $\exists$-Quantor gibt es in Java keine Entsprechung! Man kann natürlich eine andere Prozedur `undivisible(i,c)` einführen, wobei deren Korrektheit natürlich ebenso fraglich wäre wie die unseres ursprünglichen Codes:

```
public static boolean divisible(int i, int c) {
        for (int j = 2; j < i; j++) {
                if (c % j == 0) {
                        return true;
                }
        }
        return false;
}
```

Hiermit können wir nun den Code annotieren, wobei wir genau der Beweisskizze folgen:

```
public static boolean checkPrime(int c) {
        assert (c > 1) : "Vorbedingung";
        assert (false == divisible(2,c) && 2 <= c && 2 > 0);
        boolean d = false;
        assert (d == divisible(2,c) && 2 <= c && 2 > 0);
        int i = 2;
        assert (d == divisible(i,c) && i <= c && i > 0);
        while (i < c) {
                assert ((d == divisible(i,c)) && i <= c && i > 0 && i < c);
                assert (((d | (c % i == 0)) == divisible(i+1,c)) && i+1 <= c && i+1 > 0);
                d = d | (c % i == 0);
                assert ((d == divisible(i+1,c)) && i+1 <= c && i+1 > 0);
```

```
            i = i+1;
            assert ((d == divisible(i,c)) && i <= c && i > 0);
        }
        assert ((d == divisible(i,c)) && i <= c && i > 0 && i >= c);
        assert (d == divisible(c,c));
        return d;
    }
```

Ein solches Programm kann die Korrektheit der Implementierung nur überprüfen, aber nicht garantieren (nicht zuletzt haben wir `divisible` ja selbst geschrieben).