Wiederholung:

Wir haben partielle Korrektheitsaussagen,
kurz Hoare-Tripel, eingeführt:

$$\{A\}\ c\ \{B\}.$$

A: Vorbedingung, c: Programm, B: Nachbedingung

Das Hoare-Tripel ist gültig, geschrieben $\models \{A\}\ c\ \{B\}$,
falls
für alle Zustände σ, die A erfüllen,
und alle Zustände σ',
die bei Terminierung von c auf σ erreicht werden,
gilt: σ' erfüllt B.

Als Formel:

$$\forall \sigma, \sigma' \in \text{State}: \quad \sigma \models A \wedge (c, \sigma) \Downarrow \sigma' \quad \Rightarrow \quad \sigma' \models B.$$

Für die Automatisierung ist Gültigkeit problematisch.
Der Begriff spricht explizit
über einzelne Konfigurationen (und ∞-viele davon)
und das Terminierungsverhalten von c.

Frage:
Wie soll man algorithmisch damit umgehen?

Antwort:
Induktiv! Nutze ein Beweissystem,
dessen ableitbare Theoreme genau
die gültigen Hoare-Tripel sind.

Induktiv, weil man entlang der Struktur von Programmen ableitet.

Vorteile:

- Indem man direkt über Hoare-Tripel spricht,
  hat man nicht mehr einfache Konfigurationen in der Hand.
- Indem man die Programme entlang der Struktur aufbaut,
  muss man das Terminierungsverhalten
  im Grunde nur für Befehle verstehen.

Beispielregeln:

(ASSIGN) $\dfrac{}{\{A[x/a]\}\ x := a\ \{A\}}$

(WHILE) $\dfrac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \text{while } b \text{ do } c \text{ od}\ \{A \wedge \neg b\}}$

(CONSEQUENCE) $\dfrac{A \to A' \quad \{A'\}\ c\ \{B'\} \quad B' \to B}{\{A\}\ c\ \{B\}}$

Bemerkungen:

- Die Regeln sind Syntax-gesteuert,
  bei der Anlegung eines Programms gibt es keine Mehrdeutigkeiten.
- Die Regel für Zuweisungen stimmt.
  Betrachte $x := x+2$.
  Welche Vorbedingung benötigen wir, um in Anschluss $x > 2$ zu erhalten
  Die Regel sagt:
  $$\{x > 2\ [x/x+2]\}\ x := x+2\ \{x > 2\}.$$
  Tatsächlich:
  $$x > 2\ [x/x+2] \equiv x+2 > 2 \iff x > 0.$$

- In (WHILE) nennt man I die Schleifeninvariante.
  Warum Invariante?
  Weil die Prämisse fordert, dass volle Ausführungen
  des Schleifenrumpfs I erhalten.
  Die Schleifenausführung ist nur möglich, wenn b gilt.
  Sollte b also fehlschlagen, terminiert die Schleife
  mit Nachbedingung $\neg b$.
  Sollte die Schleife divergieren, darf man $\neg b$
  einfach als Nachbedingung annehmen.
  Daher ist die Regel, zumindest intuitiv, sound.
- Regel (CONSEQUENCE) nutzt gültige Implikationen
  $\dots$ der Prämisse.
  Implikationen als gültig nachzuweisen, ist schwer bis unentscheidbar.
  Zum Glück benutzen normale Programme keine tiefe Mathematik.
  und der Nachweis der Implikationen ist in der Praxis leicht.
  Intuitiv ist (CONSEQUENCE) sound, da wir
  die Vorbedingung verstärken (so dass sie von weniger Zuständen
  erfüllt wird)
  und die Nachbedingung abschwächen (so dass mehr Zustände sie erfüllen).
  Die Regel kann verstanden werden
  als Interface zwischen Programmverifikation und Logik.
- Schleifeninvarianten finden ist schwierig!
  Wie wir sehen werden, lässt sich die Programmverifikation vollständig automatisieren,
  bis auf das Finden von Schleifeninvarianten und
  unter der Annahme, dass logische Reasoning kann automatisiert werden.

Beispiel (Fakultät):

Betrachte das Programm

$$c \equiv \underline{\text{while}}\ x > 0\ \underline{\text{do}}$$
$$y := y * x;$$
$$x := x - 1;$$
$$\underline{\text{od}}$$

Wir möchten zeigen, dass c die Fakultätsfunktion berechnet:

$$n! := n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1 \quad \text{mit} \quad 0! := 1.$$

Genauer wollen wir zeigen:

$$\vdash \{x = n \wedge n \geq 0 \wedge y = 1\}\ c\ \{y = n!\},$$

was mit Soundness des Hoare-Kalküls bedeutet, dass das Tripel gültig ist.

- Da c eine while-Schleife ist, benötigen wir die Regel (WHILE). Um diese Regel zu verwenden (oder anzuwenden), benötige Schleifeninvariante:

  Wähle $I \equiv y * x! = n! \wedge x \geq 0$.

- Um zu zeigen, dass es sich bei I tatsächlich um eine Schleifeninvariante handelt, zeige:

  $$\vdash \{I \wedge x > 0\}\ y := y * x;\ x := x - 1;\ \{I\}.$$

  Beginne hinten und wende (ASSIGN) an:

  $$\vdash \{I[x/x-1]\}\ x := x - 1\ \{I\}$$

Wende (ASSIGN) ein weiteres Mal an
und erhalte:

$$\vdash \{I[x/x-1][y/y*x]\}\ y := y*x\ \{I[x/x-1]\}$$

Auf diese beiden Anwendungen von (ASSIGN)
ist (SEQ) anwendbar und liefert:

$$\vdash \{I[x/x-1][y/y*x]\}\ y := y*x;\ x := x-1\ \{I\}.$$

Daraus wollen wir nun das gesuchte

$$\vdash \{I \wedge x>0\}\ y := y*x;\ x := x-1\ \{I\}$$

mit (CONSEQUENCE) ableiten.
Dazu benötigen wir Gültigkeit der Implikation

$$I \wedge x>0 \rightarrow I[x/x-1][y/y*x].$$

Es gilt:

$$I[x/x-1][y/y*x] \equiv (y*(x-1)! = n! \wedge x-1 \geq 0)[y/y*x]$$
$$\equiv y*x*(x-1)! = n! \wedge x-1 \geq 0.$$

Ferner gilt:

$$I \wedge x>0 \equiv y*x! = n! \wedge x \geq 0 \wedge x>0$$
$$\Leftrightarrow y*x*(x-1)! = n! \wedge x \geq 1$$
$$\Leftrightarrow I[x/x-1][y/y*x].$$

- Die Anwendung von (WHILE) liefert:

$$\vdash \{I\}\ c\ \{I \wedge \neg x>0\}.$$

Um daraus die gewünschte Aussage abzuleiten,
s. nutze wieder (CONSEQUENCE).

Es gilt:

$$x = n \wedge n \geq 0 \wedge y = 1 \Rightarrow y * x! = n! \wedge x \geq 0 \equiv I.$$

Ferner gilt:

$$I \wedge \neg(x > 0) \Rightarrow y * x! = n! \wedge x \geq 0 \wedge \neg(x > 0)$$

$$(\Leftarrow) \Rightarrow y * x! = n! \wedge x = 0$$

$$\Rightarrow y * 0! = n!$$

$$\Rightarrow y = n!$$

Damit lässt sich die Vorbedingung des obigen Hoare-Tripels verstärken und die Nachbedingung abschwächen.

Mit (CONSEQUENCE) erhalten wir also:

$$\vdash \{x = n \wedge n \geq 0 \wedge y = 1\}\ c\ \{y = n!\}.$$

Beweisbaum:

Nehme folgende Abkürzungen:

$A \equiv x = n \wedge n \geq 0 \wedge y = 1$

$B \equiv y = n!$

$S \equiv I[x/x-1][y/y*x]$

body := $y := y * x;\ x := x - 1$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{S\}\ y := y*x\ \{I[x/x-1]\}}(\text{ASSIGN})\quad \dfrac{}{\{I[x/x-1]\}\ x := x-1\ \{I\}}(\text{ASSIGN})}{\{S\}\ \text{body}\ \{I\}}(\text{SEQ})}{I \wedge x > 0 \rightarrow S \quad \{S\}\ \text{body}\ \{I\} \quad I \rightarrow I}(\text{CONSEQUENCE})}{\{I \wedge x > 0\}\ \text{body}\ \{I\}}(\text{WHILE})$$

$$\dfrac{A \rightarrow I \quad \{I\}\ c\ \{I \wedge \neg(x > 0)\} \quad I \wedge \neg(x > 0) \rightarrow B}{\{A\}\ c\ \{B\}}(\text{CONSEQUENCE})$$