4.2.3 Verification Conditions

Ziel: Zeige $\models \{A\}\, c\, \{B\}$ (vollständig) automatisch.

Idee:
- Wir haben bisher gezeigt, dass $\models \{A\}\, c\, \{B\}$ äquivalent ist zur Validität von $A \to \underbrace{pred(c, B)}_{\hat{=} wlp}$ (weakest liberal precondition)
- Beachte, dass die Formel $A \to pred(c,B)$ „reine Logik" ist: die Formel ist frei von Programm c.
- Wir hoffen, dass Solver solche Formeln (deren Validität) automatisch zeigen können. | Problem: Schleifeninvariante

  ↳ In den letzten Jahren wurden Solver immer schneller und mächtiger; sowohl in den entscheidbaren, als auch in den unentscheidbaren Fragmenten (von FOL)

Ansatz: „Programmier-Overhead"
- Erlaube/Erzwinge gewisse Annotationen im Program
  - ↳ zwischendurch
  - ↳ ~~loop~~ Schleifen Invariante
    - ↳ z.B. zur schnelleren Konvergenz des Verfahrens; besseres Verständnis; ...
- Vereinfache damit die Aufgabe des Solvers / der Automatisierung

Definition: (Annotierte While-Programme)

$$c ::= skip \mid x := a \mid c_1 ; c_2 \mid c_1 ; \underbrace{\{D\}}_{\text{Annotation in Sequenzen optional}} c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}$$
$$\mid \text{while } b \text{ do } \underbrace{\{D\}}_{\text{Schleifeninvariante: Pflicht!}} c_1 \text{ end}$$

wobei D eine Assertion ist.

- Annotationen in Sequenz sind optional, da sie (einfach) berechnet werden können.
- Wie bisher ~~beim~~ schreiben wir ~~Hoare~~ Hoare Triple $\{A\}\, c\, \{B\}$ für annotierte While-Programme c.
  Das Hoare Triple ist gültig, falls $\{A\}\, c'\, \{B\}$ gültig ist, wobei c' ein reguläres While-Programm ist, ~~welches c entspricht, der keine Annotationen enthält.~~ welches c ohne Annotationen entspricht.

Beispiel: (Fakultät)

```
while x>0 do
    {y·x! = n! ∧ x ≥ 0}
    y = y*x;
    x = x-1;
end
```

Ansatz in Detail:

Wir wollen jedem Hoare Triple $\{A\}\, c\, \{B\}$, wobei c ein annotiertes Programm ist, eine Menge an Zusicherungen zuordnen, deren Validität die Validität des # zugehörigen Hoare Triples implizit.
Wir nennen diese Menge an Zusicherungen Verification Conditions, geschrieben $VC(\{A\}\, c\, \{B\})$.

- Betrachte das Hoare Triple $\{A\}\, c_1; c_2\, \{B\}$, wobei c_1 und c_2 schleifenfrei sind. Dann können wir eine Annotation generieren:

$$\{A\}\, c_1\, \{D\}\, c_2\, \{B\} \quad \text{mit} \quad D = pred(c_2, B)$$

~~- Somit reduzieren wir~~
- Somit annotieren wir das gesamte Programm, und müssen nur noch die Validität der einzelnen $\{A\}\, c\, \{B\}$ prüfen, wobei c primitive Anweisungen sind.

- Problematisch sind Schleifen:
  (1) Wie generieren wir Verification Conditions für Schleifen?
  (2) Wie generieren wir Assertions für nicht-annotierte Sequenzen, die Schleifen enthalten?

ad 1) Betrachte $\{A\}$ while b do $\{D\}$ c $\{B\}$

- Dabei soll D eine Schleifeninvariante sein.
  D.h. es soll gelten:

$$\models \{D \wedge b\}\, c\, \{D\}$$

- Wenn nun zusätzlich gilt:

$$A \to D \quad \text{und} \quad D \wedge \neg b \to B$$

dann erhalten wir mit (CONSEQUENCE)

$$\models \{A\} \text{ while } b \text{ do } c \text{ end } \{B\}$$

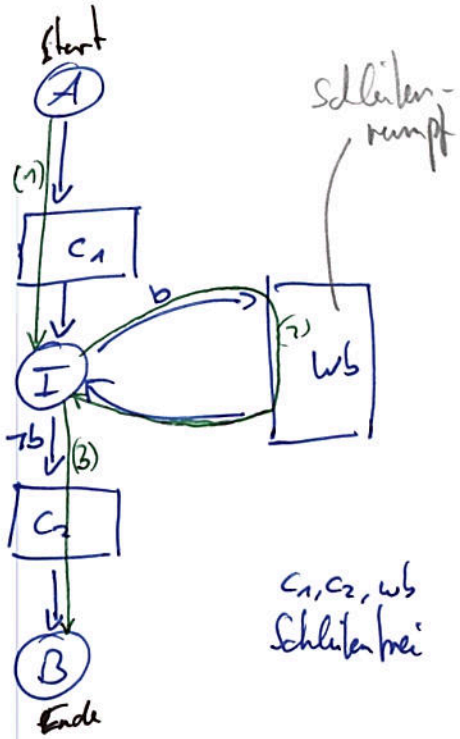
- Dementsprechend, sollte die Verification Condition für While-Schleifen wie folgt lauten:

$$vc(\{A\} \text{ while } b \text{ do } \{D\}\, c \text{ end } \{B\})$$
$$:= vc(\{D \wedge b\}\, c\, \{D\}) \cup \{A \to D\} \cup \{D \wedge \neg b \to B\}$$

ad 2) Betrachte $\{A\}\, c;w\, \{B\}$, wobei w eine While-Schleife mit Invariante I ist.

- Wir suchen ein D mit: $\models \{A\}c\{D\}w\{B\}$ impliziert $\models \{A\}c;w\{B\}$
- Wissen nach 1): falls $\models vc(\{D\}w\{B\})$, dann
  - I ist Schleifeninvariante (gilt vor/nach/während Schleife)
  - $D \rightarrow I$
- Wählen also $D := I$, da I durch c auf jeden Fall hergestellt werden muss

- Anschaulich: "Basic Paths"
  - Teile das Program in sogenannte Basic Paths auf
  - Ein Basic Path enthält keine Schleifen (keine Kreise im Kontrollflussgraphen; If-Blöcke sind erlaubt).
  - Genauer: ein Basic Path geht
    von start/Invariante
    zu Ende/Invariante



  - Im Bild sind die Basic Paths
    (1) $A - c_1 - I$
    (2) $I - wb - I$
    (3) $I - c_2 - B$
  - Wenn wir diese als Hoare Triple notieren, nehmen wir die Schleifenbedingung mit auf (vgl. Assume-Statement statt Bedingungen in WHILE/IF)

(1) $\{A\}\, c_1\, \{I\}$

(2) $\{I \wedge b\}\, wb\, \{I\}$

(3) $\{I \wedge \neg b\}\, c_2\, \{B\}$

Äquivalent:

(3') $I \wedge \neg b \rightarrow pred(c_2, B)$,
$\{pred(c_2, B)\}\, c_2\, \{B\}$

Definition

Die Funktion vc generiert zu jedem annotierten Hoare Tripel $\{A\}\, c\, \{B\}$ eine Menge an Verification Conditions wie folgt:

$$vc(\{A\}\ skip\ \{B\}) := \{A \to B\}$$

$$vc(\{A\}\ x=a\ \{B\}) := \{A \to B[x \mapsto a]\}$$

$$vc(\{A\}\ c_1; \{D\}\ c_2\ \{B\}) := vc(\{A\}\ c_1\ \{D\}) \cup vc(\{D\}\ c_2\ \{B\})$$

$$vc(\{A\}\ \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}\ \{B\}) := vc(\{A \wedge b\}\ c_1\ \{B\}) \cup vc(\{A \wedge \neg b\}\ c_2\ \{B\})$$

$$vc(\{A\}\ \text{while } b \text{ do } \{D\}\ c \text{ end}\ \{B\}) := vc(\{D \wedge b\}\ c\ \{D\}) \cup \{A \to D\} \cup \{D \wedge \neg b \to B\}$$

$$vc(\{A\}\ c_1; c_2\ \{B\}) := vc(\{A\}\ c_1\ \{pred'(c_2, B)\}\ c_2\ \{B\})$$

mit

$$pred'(c,B) = \begin{cases} D, & \text{falls } c \equiv \text{while } b \text{ do } \{D\} \text{ end} \\ pred(c,B), & \text{sonst} \end{cases}$$

Also: Finden Schleifeninvarianten und ob wlp oder Basic Paths hergestellt wird

Satz (Soundness / Korrektheit)

$\models vc(\{A\}\, c\, \{B\})$ impliziert $\models \{A\}\, c\, \{B\}$

Validität auf Formeln — Validität auf Hoare Tripeln

Beweis: Induktion nach der Struktur von Programmen. ✓

Beispiel:

$\omega \equiv$ while $x>0$ do

$\{I\} \equiv \{ y \cdot x! = n! \wedge x \geq 0 \}$

$y = y \cdot x;$

$x = x - 1;$

end

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

$B \equiv y = n!$

Dann liefert vc folgende Verifikationsbedingungen für Assertions A, B:

$vc(\{A\}\,\omega\,\{B\})$

$= vc(\{I \wedge x>0\}\ y=y\cdot x;\ x=x-1\ \{I\}) \cup \{A \to I\} \cup \{I \wedge x \leq 0 \to B\}$

$= vc(\{I \wedge x>0\}\ y=y\cdot x\ \{I[x \mapsto x-1]\ |\ x=x-1\ \{I\}) \cup \{A \to I,\ I \wedge x \leq 0 \to B\}$

$= vc(\{I \wedge x>0\}\ y=y\cdot x\ \{I[x \mapsto x-1]\}) \cup vc(\{I[x \mapsto x-1]\}\ x=x-1\ \{I\}) \cup \{A \to I,\ I \wedge x \leq 0 \to B\}$

$= \underbrace{\{I \wedge x>0 \to I[x \mapsto x-1][y \mapsto y \cdot x]\}}_{V_1} \cup \underbrace{\{I[x \mapsto x-1] \to I[x \mapsto x-1]\}}_{V_2} \cup \underbrace{\{A \to I\}}_{V_3} \cup \underbrace{\{I \wedge x \leq 0 \to B\}}_{V_4}$

Es gilt:

- $\vDash V_1$
- $\vDash V_3$
- $\vDash V_4$

} siehe Beispiel 4.14

- $\vDash V_2$ } trivialer Weise

~~Zusammen also (nach Satz):~~

~~$\vDash \{A\}\,\omega\,\{B\}$~~

Zusammen also:

$\vDash vc(\{A\}\,\omega\,\{B\})$

Nach Satz Korrektheit also:

$\vDash \{A\}\,\omega\,\{B\}$

Vorteil von Verifikationsbedingungen:

Validität von $V_1, \ldots, V_4$ kann von Theorem Beweisern, SAT Solvern, SMT Solvern, ... bewiesen werden.

Das Prüfen der Verifikationsbedingungen ist somit automatisierbar

z.B. Microsoft Z3: rise4fun.com/z3

Lemma:

$\models \{A\}\, c\, \{B\}$ folgt nicht unbedingt $\models vc(\{A\}\, c\, \{B\})$

Beweis:

Betrachte $\models \{true\}$ while false do $\{false\}$ skip end $\{true\}$.

Es gilt:

$vc(\{true\}$ while false do $\{false\}$ skip end $\{true\})$

$= vc(\{\underbrace{false}_{Inv} \wedge \underbrace{false}_{Cond}\}$ skip $\{\underbrace{false}_{Inv}\} \cup \{\underbrace{true}_{pre} \rightarrow \underbrace{false}_{Inv}\} \cup \{\underbrace{false}_{Inv} \wedge \underbrace{\neg false}_{\neg Cond} \rightarrow \underbrace{true}_{post}\})$

Da $true \rightarrow false$ ja nicht valide ist, also $\not\models true \rightarrow false$, gilt: $\not\models vc(\ldots)$, die generierten Verifikationsbedingungen sind nicht valide, obwohl das zu Grunde liegende Hoare Tripel valide ist. □

Bemerkung zu Funktionen

- Funktionen können ähnlich zu Schleifen behandelt werden
- Deren Form Spezifikation von Funktionen ist: pre/post condition
- Konstruiere Basic Path so, dass sie nicht über solche pre/post conditions reichen
- Also intuitiv: ersetze Funktionsaufrufe durch Spezifikation, und prüfe ob Funktionen ihre Spezifikation einhalten
- Beachte: man muss einen Mechanismus implementieren, der es erlaubt die Änderung der globalen Variablen zu berücksichtigen.

Siehe Bradley/Manna „The Calculus of Computation"