9. Rely-Guarantee

RG: PhD Cliff Jones 1981

RGSep: PhD Viktor Vafeiadis 2007

Ziel: Korrektheitsbeweise für parallele Programme mit Shared-Variables.

Problem: Berechnungen des einen Threads ändern die gemeinsamen Variablen und beeinflussen damit die Berechnungen des anderen Threads.

Ideen: (1) Beschränke die Einflüsse mittels atomic (-)

↳ Concurrent-Separation-Logic

↳ Begrenzt anwendbar.

(2) Integriere die (wechselseitigen) Einflüsse in den Beweis

:) Fokus auf Beweise anstatt feingranulare Berechnungen.

Ansatz:
- Beweise die Korrektheit eines jeden Threads in Isolation.
- Berücksichtige dabei die Einflüsse / Interferences des Partner-Threads → Rely
- Ermittle die eigenen Interferences für den Partner-Thread → Guarantee.
- Komponiere die Thread-lokalen Beweise: die Garantien des einen Threads müssen abgedeckt sein von den Relies des anderen.

9.1 Einleitung

Ziel: Skizzire eine Rely-Guarantee-Programmlogik für C-- (hier Heap).

Idee: Spezifikationen sind keine Hoare-Tupel sondern 5-Tupel

Definition:

Spezifikationen haben die Form:

$$c : (A, R, G, B)$$

A: Vorbedingung, R: Rely, G: Guarantee, B: Nachbedingung.

Dabei gilt:

- A, B: Assertions über einzelnen States, also $[\![A]\!], [\![B]\!] \subseteq State$
- R, G: Assertions über Paaren von States (Pre- und Poststate), also $[\![R]\!], [\![G]\!] \subseteq State \times State$ (Relationen zwischen States).
- A, R: Annahmen (Premises)
- G, B: Schlüsse (Conclusions).

Definition:

$c : (A, R, G, B)$ gilt, falls $\forall n \in \mathbb{N}. \forall s \in State$ mit $[\![A]\!] s = true$:

$safe(c, s, R, G, B)$.

Dabei ist safety definiert als:

- $safe_0(c, s, R, G, B) := true$
- $safe_{n+1}(c, s, R, G, B) := (1) \wedge (2) \wedge (3)$ mit

(1) $c = skip \Rightarrow [\![B]\!] s = true$.

(2) $\forall c', \forall s'. (c, s) \to (c', s') \Rightarrow$

$[\![G]\!] s, s' = true \wedge safe_n(c', s', R, G, B)$.

(3) $\forall s''.\ [\![R]\!]\, s, s'' = \text{true} \Rightarrow \text{safe}(c, s'', R, G, B)$.

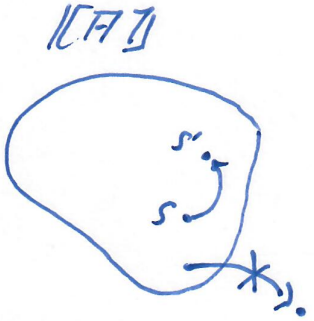
Bemerkung:

(1) Bei Termination gilt B.

(2) Eigene Änderungen des Shared-State sind in G gefasst.

(3) Änderungen des Shared-State durch andere Threads invalidieren Safety nicht.

Der Begriff der Stabilität sagt aus, dass eine Relation zwischen Zuständen eine Aussage nicht invalidieren kann.



Definition:

Sei A eine Aussage über States und R eine Aussage über Paaren von States.

Dann heißt A stabil unter R, falls

$$\forall s, s' \in \text{State}.\ [\![A]\!]\, s \wedge [\![R]\!]\, s, s' \Rightarrow [\![A]\!]\, s'.$$

Beweisregeln:

$$\text{(BASIC)}\ \frac{\begin{array}{l}\{A \wedge s_0 = s\}\ c\ \{B \wedge G(s_0, s)\} \quad \text{// Hoare-Tripel gültig} \\ A \text{ stabil unter } R \qquad c \text{ atomarer Befehl} \\ B \text{ stabil unter } R\end{array}}{c : (A, R, G, B)}$$

Wenn initial $s_0 = s$ gilt und s gemerkt wird, gilt im Nachlauf $G(s_0, s)$.

→

$$(SEQ)\quad \frac{c_1 : (A, R, G, B) \qquad c_2 : (B, R, G, C)}{c_1 ; c_2 : (A, R, G, C)}$$

$$(PAR)\quad \frac{c_1 : (A, R \cup G_2, G_1, B_1) \qquad c_2 : (A, R \cup G_1, G_2, B_2)}{c_1 \| c_2 : (A, R, G_1 \cup G_2, B_1 \wedge B_2)}$$

$R \cup G_2$: Das Environment von Thread c_1 ist
- c_2 und
- das Environment der Parallelkomposition (A)

$G_1 \cup G_2$: Jeder Schritt des Programms $c_1 \| c_2$
kommt von c_1 oder von c_2
und erfüllt damit G_1 oder G_2.

$B_1 \wedge B_2$: Ab der Terminierung von c_1 gilt B_2.
Ab der Terminierung von c_2 gilt B_1.
Also gilt bei Terminierung von $c_1 \| c_2$ $B_1 \wedge B_2$.

$$(CONSEQ)\quad \frac{c : (A, R, G, B) \qquad A' \Rightarrow A \qquad B \Rightarrow B' \qquad R' \Rightarrow R \qquad G \Rightarrow G'}{c : (A', R', G', B')}$$

Man kann die Annahmen verstärken
und die Schlüsse abschwächen.

Beispiel:

Definiere

$G_1(x, a, b, x', a', b') := x' \geq x \wedge a' = a \wedge b' \leq b$

$G_2(x, a, b, x', a', b') := x' = x$

// Keine Aussagen über a', a, b', b.

$R :=$ false.

$\{true\}$

$\{true\}$ $a := x;$ $\{a \leq x\}$ (1)

$\{true\}$ $b := x;$ $\{a \leq b\}$ (2)

(3) $\{true\}$ $x := x + 1;$ $\{true\}$

$\{a \leq b\}$

Erinnerung

(INVBASIC)

$$\frac{\{A \wedge s_0 = s\}\ c\ \{B \wedge G(s_0, s)\} \qquad A \text{ stabil unter } R \qquad c \text{ atomar} \qquad B \text{ stabil unter } R}{c : (A, R, G, B)}$$

(1) Zeige: $a := x : (true, R \vee G_2, G_2, a \leq x)$

Prüfe die Gültigkeit des Hoare-Tripels:

$\{true \wedge x_0 = x \wedge a_0 = a \wedge b_0 = b\}$

$a := x;$

$\{a \leq x \wedge \underbrace{x = x_0}_{G_2(x_0, a_0, b_0, x, a, b)}\}$ ✓

Außerdem muss Stabilität geprüft werden:

(i) $\models \text{true} \wedge \underbrace{(R \vee G_1)}(x, a, b, x', a', b') \longrightarrow \text{true}$.

(ii) $\models a \le x \wedge x' \ge x \wedge a' = a \wedge b' = b \longrightarrow a' \le x'$.

Da hier Allgemeingültigkeit geprüft wird,
werden alle Belegungen der Variablen berücksichtigt.
Das entspricht dem Allquantor.

(2) Zeige: $b := x : (a \le x,\ R \vee G_1,\ G_2,\ a \le b)$.

Prüfe die Gültigkeit des Hoare-Tripels:

$$\{\overbrace{x_0 = x \wedge a_0 = a \wedge b_0 = b}^{S_0 = S} \wedge \overbrace{a \le x}^{A}\}$$

$$b := x$$

$$\{\underbrace{a \le b}_{B} \wedge \underbrace{x = x_0}_{G_2}\}$$ ✓

Auch hier muss Stabilität geprüft werden:

(i) ✓

(ii) ✓

(iii) $\models a \le b \wedge x' \ge x \wedge a' = a \wedge b' = b \longrightarrow a' \le b'$. ✓

Beachte:
Sowohl für die Prüfung des Hoare-Tripels
als auch für die Stabilitätschecks
nutzen wir einen Solver.

(3) Zeige: $x := x+1 : (true, R \vee G_2, G_2, true)$

Prüfe die Gültigkeit des Hoare-Tripels:

$\{ true \wedge x_0 = x \wedge a_0 = a \wedge b_0 = b \}$

$x := x+1;$

$\{ \underbrace{true}_{\beta} \wedge \underbrace{x \geq x_0 \wedge a = a_0 \wedge b = b_0}_{G_1(x_0, a_0, b_0, x, a, b)} \}$ ✓

Prüfe Stabilität:

(iv) $\models true \wedge (R \vee G_2)(x, a, b, x', a', b') \rightarrow true.$ ✓

(v) $\models true \wedge (R \vee G_2) \rightarrow true.$

Es gilt:

$x := x+1 : (true, R \vee G_2, G_2, true)$

$a := x; b := x : (true, R \vee G_2, G_2, a \leq b)$

$x := x+1 \;\|\; \begin{matrix} a := x; \\ b := x. \end{matrix} \; (true, R, G_1 \vee G_2, true \wedge a \leq b).$