

Verificarea programelor

`https://tinyurl.com/cursVVS`

Să reluăm un exemplu

```
// assume(n>2);  
void partition(int a[], int n) {  
    int pivot = a[0];  
    int lo = 1, hi = n-1;  
    while (lo <= hi) {  
        while (lo < n && a[lo] <= pivot)  
            lo++;  
        while (a[hi] > pivot)  
            hi--;  
        if (lo < hi)  
            swap(a,lo,hi);  
    }  
}
```

Cum putem raționa despre acest (fragment de) program ?

Începuturile verificării programelor

Scopul: formalizarea semanticii limbajelor de programare

Robert W. Floyd. *Assigning Meanings to Programs* (1967)

"an adequate basis for formal definitions of the meanings of programs [...] in such a way that a rigorous standard is established for proofs"

"If the initial values of the program variables satisfy the relation R_1 , the final values on completion will satisfy the relation R_2 ."

Floyd: Assigning Meanings to Programs

Metoda: anotarea unui program (schemă logică) cu aserțiuni

verification condition: o formulă $V_c(P; Q)$ a.î.

dacă P e adevărat *înainte* de a executa c ,
atunci Q e adevărat *la ieșire*.

strongest verifiable consequent (pt. un program + o condiție inițială)
= cea mai puternică proprietate valabilă *după* execuția programului

Formulele/asertiunile: exprimate în *logica predicatelor de ordinul I*

Floyd: Assigning Meanings to Programs

Lucrarea lui Floyd:

dezvoltă reguli generale pt. combinarea condițiilor de verificare
și reguli specifice pentru diferitele tipuri de instrucțiuni

introduce *invariantii* pentru raționamentele despre cicluri

tratează *terminarea* folosind o măsură pozitivă descrescătoare

Lucrările lui Hoare

C.A.R. Hoare. *An Axiomatic Basis for Computer Programming* (1969)

- ca Floyd, tratează *precondiții* și *postcondiții* pentru execuția unei instrucțiuni
- notația de *triplet Hoare* pune mai clar în evidență relația dintre instrucțiune și cele două aserțiuni
- lucrează cu programe textuale, nu scheme logice

Lucrările lui Hoare

– Notăție: *corectitudine parțială* $\{P\} S \{Q\}$

Dacă S e executat într-o stare care satisface P ,
și execuția lui S se termină
 $\Rightarrow$ starea rezultantă satisface Q

Lucrările lui Hoare

- Notăție: *corectitudine parțială* $\{P\} S \{Q\}$

Dacă S e executat într-o stare care satisface P ,
și execuția lui S se termină
 $\Rightarrow$ starea rezultantă satisface Q

- Ulterior: raționamente similare pt. *corectitudine totală* $[P] S [Q]$

Dacă S e executat într-o stare care satisface P
 $\Rightarrow$ execuția lui S se termină
și starea rezultantă satisface Q

Lucrările lui Hoare

- Notăție: *corectitudine parțială* $\{P\} S \{Q\}$

Dacă S e executat într-o stare care satisface P ,
și execuția lui S se termină
 $\Rightarrow$ starea rezultantă satisface Q

- Ulterior: raționamente similare pt. *corectitudine totală* $[P] S [Q]$

Dacă S e executat într-o stare care satisface P
 $\Rightarrow$ execuția lui S se termină
și starea rezultantă satisface Q

Aplicație riguroasă: C.A.R. Hoare. *Proof of a Program: FIND* (1971)

Axiomele (regulile) lui Hoare

Sunt definite pentru fiecare tip de instrucțiune în parte;
prin combinația lor, se poate raționa despre programe întregi

Axiomele (regulile) lui Hoare

Sunt definite pentru fiecare tip de instrucțiune în parte;
prin combinația lor, se poate raționa despre programe întregi

Atribuire: $\frac{}{\{Q[x/E]\} \ x := E \ \{Q\}}$ unde $Q[x/E]$ e substituția lui x cu E în Q

Ex: $\{x = y - 2\} \ x := x + 2 \ \{x = y\}$

(în rezultat $x = y$, substituim x cu expresia atribuită, $x + 2$
și obținem $x + 2 = y$, deci $x = y - 2$)

Obs: scrierea “înapoi” a regulii (P în funcție de Q) simplifică exprimarea

Axiomele (regulile) lui Hoare

Secvențiere:

$$\frac{\{P\} S_1 \{Q\} \quad \{Q\} S_2 \{R\}}{\{P\} S_1; S_2 \{R\}}$$

Decizie:

$$\frac{\{P \wedge E\} S_1 \{Q\} \quad \{P \wedge \neg E\} S_2 \{Q\}}{\{P\} \text{ if } E \text{ then } S_1 \text{ else } S_2 \{Q\}}$$

Regulile lui Hoare (cont.)

Ciclul cu test (inițial): cheia în raționamentul despre programe

- trebuie găsit un *invariant* I = o proprietate menținută adevărată de fiecare execuție a ciclului (exprimată în punctul dintre iterații)

Regulile lui Hoare (cont.)

Ciclul cu test (inițial): cheia în raționamentul despre programe

- trebuie găsit un *invariant* I = o proprietate menținută adevărată de fiecare execuție a ciclului (exprimată în punctul dintre iterații)
- dacă intrăm în ciclu (E), invariantul e menținut după o iterație S
- dacă nu mai intrăm ($\neg E$), invariantul implică concluzia Q

Regulile lui Hoare (cont.)

Ciclul cu test (inițial): cheia în raționamentul despre programe

- trebuie găsit un *invariant* I = o proprietate menținută adevărată de fiecare execuție a ciclului (exprimată în punctul dintre iterații)
- dacă intrăm în ciclu (E), invariantul e menținut după o iterație S
- dacă nu mai intrăm ($\neg E$), invariantul implică concluzia Q

Regula lui Hoare pentru while

$$\frac{\{I \wedge E\} S \{I\} \quad \{I \wedge \neg E\} \Rightarrow \{Q\}}{\{I\} \text{ while } E \text{ do } S \{Q\}}$$

Exemplu de aplicare a regulilor lui Hoare

Găsim pe n știind că inițial e între lo și hi :

```
while (lo < hi) {    // căutare binară; I: lo <= n && n <= hi
    m = (lo + hi) / 2;
    if (n > m)        // ambele cazuri mențin lo<=n && n<=hi
        lo = m+1;    // n > m => n >= m+1 => n >= lo
    else hi = m;      // !(n > m) => n <= m => n <= hi
}                    // I rămâne adevărat
// lo<=n && n<=hi && !(lo<hi) => lo==n && n==hi
assert(n == lo && n == hi);
```


Regulile lui Hoare în prezența pointerilor

Să stabilim $\{P\} \ *x = 2 \ \{v + *x = 4\}$

Regulile lui Hoare în prezența pointerilor

Să stabilim $\{P\} \quad *x = 2 \quad \{v + *x = 4\}$

Care este condiția P ?

Răspunsul corect:

$$v = 2 \vee x = \&v$$

Dar aplicarea regulii $(v + *x = 4)[*x/2]$ pierde cazul al doilea.

Regulile lui Hoare în prezența pointerilor

Trebuie să modelăm memoria

Regulile lui Hoare în prezența pointerilor

Trebuie să modelăm memoria: m = memorie, a = adresă, d = dată.

Fie funcțiile $rd(m, a)$ return d și $wr(m, a, d)$ return m'

Regulile lui Hoare în prezența pointerilor

Trebuie să modelăm memoria: m = memorie, a = adresă, d = dată.

Fie funcțiile $rd(m, a)$ return d și $wr(m, a, d)$ return m'

Avem regula: $rd(wr(m, a_1, d), a_2) = \begin{cases} d & \text{dacă } a_2 = a_1 \\ rd(m, a_2) & \text{dacă } a_2 \neq a_1 \end{cases}$

Regulile lui Hoare în prezența pointerilor

Trebuie să modelăm memoria: m = memorie, a = adresă, d = dată.

Fie funcțiile $rd(m, a)$ return d și $wr(m, a, d)$ return m'

Avem regula: $rd(wr(m, a_1, d), a_2) = \begin{cases} d & \text{dacă } a_2 = a_1 \\ rd(m, a_2) & \text{dacă } a_2 \neq a_1 \end{cases}$

Trebuie să deducem o proprietate a memoriei m din relația:

$$rd(wr(m, x, 2), \&v) + rd(wr(m, x, 2), x) = 4$$

Regulile lui Hoare în prezența pointerilor

Trebuie să modelăm memoria: m = memorie, a = adresă, d = dată.

Fie funcțiile $rd(m, a)$ return d și $wr(m, a, d)$ return m'

Avem regula: $rd(wr(m, a_1, d), a_2) = \begin{cases} d & \text{dacă } a_2 = a_1 \\ rd(m, a_2) & \text{dacă } a_2 \neq a_1 \end{cases}$

Trebuie să deducem o proprietate a memoriei m din relația:

$$rd(wr(m, x, 2), \&v) + rd(wr(m, x, 2), x) = 4$$

$$rd(wr(m, x, 2), \&v) + 2 = 4$$

$$rd(wr(m, x, 2), \&v) = 2$$

$$x = \&v \wedge 2 = 2 \vee x \neq \&v \wedge rd(m, \&v) = 2$$

$$x = \&v \vee v = 2$$

Operatorul *weakest precondition* al lui Dijkstra

E.W. Dijkstra. *Guarded Commands, Nondeterminacy and Formal Derivation of Programs* (1975)

- pentru o instrucțiune S și postcondiție dată Q pot exista *mai multe* precondiții P astfel încât $\{P\} S \{Q\}$ sau $[P] S [Q]$.
- Dijkstra calculează o precondiție *necesară și suficientă* $wp(S, Q)$ pentru terminarea cu succes a lui S cu postcondiția Q .

Operatorul *weakest precondition* al lui Dijkstra

E.W. Dijkstra. *Guarded Commands, Nondeterminacy and Formal Derivation of Programs* (1975)

- pentru o instrucțiune S și postcondiție dată Q pot exista *mai multe* precondiții P astfel încât $\{P\} S \{Q\}$ sau $[P] S [Q]$.
- Dijkstra calculează o precondiție *necesară și suficientă* $wp(S, Q)$ pentru terminarea cu succes a lui S cu postcondiția Q .
- necesară (*weakest*): dacă $[P] S [Q]$ atunci $P \Rightarrow wp(S, Q)$
- wp e un *transformator de predicate* (transformă post- în precondiție)
- permite definirea unui *calcul* cu astfel de transformări

Precondițiile lui Dijkstra (cont.)

Atribuire: $wp(x := E, Q) = Q[x/E]$ (v. regula lui Hoare)

Secvențiere: $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$

Condițional:

$$\begin{aligned} &wp(\text{if } E \text{ then } S_1 \text{ else } S_2, Q) \\ &= (E \Rightarrow wp(S_1, Q)) \wedge (\neg E \Rightarrow wp(S_2, Q)) \end{aligned}$$

Precondițiile lui Dijkstra (cont.)

Pentru iterație e nevoie de un calcul recursiv.

Definim wp_k , în ipoteza că bucla se termină în cel mult k iterații:

$$wp_0(\text{while } E \text{ do } S, Q) = \neg E \Rightarrow Q \quad (\text{nu se intră în ciclu})$$

$$\begin{aligned} wp_{k+1}(\text{while } E \text{ do } S, Q) \\ = (E \Rightarrow wp(S, wp_k(\text{while } E \text{ do } S, Q))) \wedge (\neg E \Rightarrow Q) \end{aligned}$$

Precondițiile lui Dijkstra (cont.)

Pentru iterație e nevoie de un calcul recursiv.

Definim wp_k , în ipoteza că bucla se termină în cel mult k iterații:

$$wp_0(\text{while } E \text{ do } S, Q) = \neg E \Rightarrow Q \quad (\text{nu se intră în ciclu})$$

$$\begin{aligned} wp_{k+1}(\text{while } E \text{ do } S, Q) \\ = (E \Rightarrow wp(S, wp_k(\text{while } E \text{ do } S, Q))) \wedge (\neg E \Rightarrow Q) \end{aligned}$$

($\leq k + 1$ iterații $\Leftrightarrow$ o iterație urmată de $\leq k$, sau 0 iterații;
echivalent cu descompunerea primului `while` în `if`)

$\Rightarrow$ se poate scrie ca formulă de punct fix

Recapitulare: verificarea prin demonstrare de teoreme

1. Se scriu triplete Hoare / precondiții Dijkstra
2. Se verifică înlănțuirea lor
(cu un demonstrator de teoreme / procedură de decizie).

Exemple:

cu regula lui Hoare pentru secvențiere

se verifică $Pre \Rightarrow wp(Prog, Post)$

se verifică $I \wedge E \Rightarrow wp(CorpCiclu, I)$ pentru cicluri