Demonstrează-mi!

Marius Minea

17 martie 2010

E corectă sortarea ?

```
void sort1(int a[], unsigned n)
{
  for (int i = 0; i < n; ++i)
    for (int j = i; ++j < n-i;) {
      if (a[i] > a[j]) {
        int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
      }
    }
}
```

Dar sortarea asta ?

```
void sort2(int a[], unsigned n)
{
  for (int i = 0; i < n; ++i)
    for (int j = i; ++j < n;) {
      if (a[i] > a[j]) {
        int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
      }
    }
}
```

Dar varianta asta ?

```
void sort3(int a[], unsigned n)
{
  for (int i = n; --i >= 0;)
    for (int j = i; --j >= 0;) {
      if (a[i] < a[j]) {
        int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
      }
    }
}
```

E corectă căutarea binară ?

```
int bsearch1(int v, int a[], unsigned n)
{
  unsigned lo = 0, hi = n-1, m;
  while (lo < hi) {
    m = (lo + hi) / 2;
    if (a[m] < v) lo = m;
    else hi = m;
  }
  if (a[m] == v) return m;
  else return -1;
}
```

Dar căutarea asta ?

```
int bsearch2(int v, int a[], unsigned n)
{
  unsigned lo = 0, hi = n-1, m;
  while (lo < hi) {
    m = (lo + hi) / 2;
    if (a[m] < v) lo = m + 1;
    else hi = m;
  }
  if (a[m] == v) return m;
  else return -1;
}
```

Cum știu **sigur** dacă programul e corect ?

Îl *testez* ?

- Cum ?
- Cât e deajuns ?

Îl *verific*. *Demonstrez*

Ce înseamnă asta ?

Pot s-o fac automat ?

Ce e o demonstrație ?

Informal: o înșiruire logică de afirmații

Mai precis: o înșiruire de formule, unde fiecare:

- e o *axiomă* (variantă: o *ipoteză*)
- sau rezultă din precedentele printr-o *regulă de inferență*

Modus Ponens: $$\frac{A \qquad A \to B}{B}$$

A demonstra e greu, a verifica e simplu

Axiome:
A1: $\alpha \to (\beta \to \alpha)$
A2: $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$
A3: $((\neg\beta) \to (\neg\alpha)) \to (((\neg\beta) \to \alpha) \to \beta)$

Exemplu: demonstrăm că $\varphi \to \varphi$

1. $\varphi \to ((\varphi \to \varphi) \to \varphi)$	**A1**
2. $\varphi \to ((\varphi \to \varphi) \to \varphi) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))$	**A2**
3. $(\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$	MP(1,2)
4. $\varphi \to (\varphi \to \varphi)$	**A1**
5. $\varphi \to \varphi$	MP(3,4)

A verifica e simplu:

$$\begin{array}{lll} i & A & \\ j & A \to B & \\ \hline k & B & \text{MP(i, j)} \end{array}$$

- verifică $0 < i, j < k$
- verifică că formula i (A) e un prefix al formulei j ($A \to B$)
  o simplă potrivire de șiruri!
- și că după $\to$ apare formula k (B)

$\Rightarrow$ Un verificator de demonstrații (*proof checker*) e foarte simplu
$\Rightarrow$ codul poate fi *verificat* pentru a avea încredere

Unificare

Declarațiile
`int f(int a[], unsigned n)`
si
`int f(int tab[], unsigned len)`
reprezintă aceeași funcție ?

Există ceva comun în expresiile
`5*5 - 4*x*y` si `a*a - 4*(t+2)*c` ?

Cum exprimăm asemănarea dintre două fragmente de cod ?
- pt. analiza duplicatelor
- pt. refactorizare

Noțiunea riguroasă: *unificarea*

Unificare: definiție

Lucrăm cu mulțimi de: constante, variabile, funcții, evtl. predicate

O *substituție* e o funcție de la variabile la termeni:

$$\sigma : V \to \mathcal{T}$$

$\Rightarrow$ prin extensie, transformă termeni în termeni (substituind variabilele)

Doi termeni T_1 și T_2 se pot *unifica* dacă există două substituții prin care devin la fel: $\sigma_1(T_1) = \sigma_2(T_2)$

Exemplu: $f(x, g(y), a)$ și $f(z, t, y)$ au pe $f(x, g(a), a)$
ca *cel mai general unificator* (most general unifier)

Implementare: Union-Find

– o variabilă se poate unifica cu orice termen

– o funcție se unifică doar cu aceeași funcție, dacă se pot unifica perechi argumentele
(caz particular: constanta e o funcție de zero argumente))

Aplicații: Unificarea în Prolog

Inferența automată de tipuri