Semantyka i weryfikacja programów 2025/26.
Egzamin I termin, 26/01/2026, zadanie 2 (weryfikacja)

Pracujemy w języku TINY$_{\mathcal{A}}$ nad typem danych rozszerzonym o jednoargumentową operację arytmetyczną _ `div2` dzielenia całkowitego przez 2 oraz o rodzaj *Array* i operacje:

$$\begin{array}{l} \mathit{newarr}\colon \mathit{Array} \\ \mathit{put}\colon \mathit{Array} \times \mathit{Int} \times \mathit{Int} \to \mathit{Array} \\ \mathit{get}\colon \mathit{Array} \times \mathit{Int} \to \mathit{Int} \\ \mathit{swap}\colon \mathit{Array} \times \mathit{Int} \times \mathit{Int} \to \mathit{Array} \end{array}$$

Nośnik rodzaju *Array* to zbiór funkcji (całkowitych) z liczb całkowitych w liczby całkowite,

$$|\mathcal{A}|_{\mathit{Array}} = \mathbf{Int} \to \mathbf{Int},$$

a operacje interpretowane są jako funkcje

$$\begin{array}{l} \mathit{newarr}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}} \\ \mathit{put}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}} \times |\mathcal{A}|_{\mathit{Int}} \times |\mathcal{A}|_{\mathit{Int}} \to |\mathcal{A}|_{\mathit{Array}} \\ \mathit{get}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}} \times |\mathcal{A}|_{\mathit{Int}} \to |\mathcal{A}|_{\mathit{Int}} \\ \mathit{swap}_{\mathcal{A}}\colon |\mathcal{A}|_{\mathit{Array}} \times |\mathcal{A}|_{\mathit{Int}} \times |\mathcal{A}|_{\mathit{Int}} \to |\mathcal{A}|_{\mathit{Array}} \end{array}$$

zdefiniowane następująco:

$$\begin{array}{l} \mathit{newarr}_{\mathcal{A}}(i) = 0 \\ \mathit{put}_{\mathcal{A}}(A, i, n) = A[i \mapsto n] \\ \mathit{get}_{\mathcal{A}}(A, i) = A(i) \\ \mathit{swap}_{\mathcal{A}}(A, i, j) = A[i \mapsto A(j), j \mapsto A(i)] \end{array}$$

dla wszystkich $i, j, n \in |\mathcal{A}|_{\mathit{Int}} = \mathbf{Int}$ i $A\colon \mathbf{Int} \to \mathbf{Int}$. Wyrażenie $\mathit{get}(A, e)$ będziemy jak zwykle zapisywać jako $A[e]$. Ponadto, w asercjach wykorzystujemy „predykat" $A{:}\mathit{Array}$ stwierdzajacy, że zmienna A jest rodzaju *Array* (pozostałe zmienne są rodzaju *Int*). Wyrażenia logiczne języka rozszerzamy też o oczywiste nierówności $e < e'$ (wyrażalne jako $e \leq e' \wedge e \neq e'$) oraz $e \geq e'$ (wyrażalne jako $e' \leq e$). Dla czytelności, w formułach ciągi nierówności będziemy zapisywać w skróconej postaci, np. $j \leq i \wedge i < k$ zapisując jako $j \leq i < k$.

Poniżej (na odwrocie) dany jest program w tym języku.
Do celów specyfikacji wprowadzamy „predykaty" (skróty formuł):

$$\begin{array}{l} H(A, p, r) \equiv A{:}\mathit{Array} \wedge \forall l.\, 2 * p \leq l < r \Rightarrow A[l \texttt{ div2}] \geq A[l] \\ S(A, r) \equiv A{:}\mathit{Array} \wedge \forall l, l'.\, 1 \leq l < l' < r \Rightarrow A[l] \geq A[l'] \end{array}$$

W razie potrzeby podobnie można zdefiniować dodatkowe pomocnicze predykaty.

Należy udowodnić całkowitą poprawność programu względem podanych warunków, podając:

- niezmienniki pętli programu oraz asercje pośrednie, które „kodują" dowód poprawności częściowej w logice Hoare'a; wymagane jest podanie niezmienników $\gamma_1, \gamma_2, \gamma_3$ oraz przynajmniej asercji $\alpha_1, \alpha_2, \alpha_3$ (ale podanie innych asercji z błędami może wpłynąć na ostateczną ocenę rozwiąznia).

- anotacje [**decr ... in ... wrt ...**] dla obu pętli, tak aby (w kontekście podanych niezmienników) wynikała z nich własność stopu pętli.

```
[n > 1 ∧ H(A,1,n)]
r := 2;
[                                                                    ]
while [γ1:                                                           ]
  r < n do [ decr                wrt        in             ]
    (if 2*r < n then t := 2*r else t := n;
[                                                                    ]
     m := r;
[                                                                    ]
     x := A[r];
[                                                                    ]
     k := r+1;
[                                                                    ]
     while [γ2:                                                      ]
        k < t do [ decr                wrt        in        ]
          (if A[k] > x then x := A[k]; m := k else skip;
[                                                                    ]
           k := k+1);
[α1:                                                                 ]
     if m > r then
       (A := swap(A,r,m);
[                                                                    ]
        r := r+1;
[                                                                    ]
        j := m;
[                                                                    ]
        while [γ3:                                                   ]
           2*j < n do [ decr             wrt        in        ]
            (k := 2*j;
[α2:                                                                 ]
             if k+1 < n then
               if A[j] ≥ A[k] and A[j] ≥A[k+1] then j := n
               else if A[k] ≥ A[k+1]
                    then (A := swap(A,j,k);
[α3:                                                                 ]
                          j := k)
                    else (A := swap(A,j,k+1);
[                                                                    ]
                          j := k+1)
             else if A[j] ≥ A[k] then j := n
                  else (A := swap(A,j,k);
[                                                                    ]
                        j := k)
           )
       ) else r := r+1
  )
[                                                                    ]
[S(A,n)]
```

Przydadzą się dodatkowe pomocnicze predykaty:

$$Hbut(A,p,r,k) \equiv A{:}Array \wedge (\forall l.(2*p \le l < r \wedge k \neq l \texttt{ div2}) \Rightarrow A[l \texttt{ div2}] \ge A[l]) \\ \wedge (2*p \le k \wedge 2*k < r \Rightarrow A[k \texttt{ div2}] \ge A[2*k]) \\ \wedge (2*p \le k \wedge 2*k+1 < r \Rightarrow A[k \texttt{ div2}] \ge A[2*k+1])$$

$$Max(A,p,r,k) \equiv A{:}Array \ \wedge p \le k < r \wedge \forall l.p \le l < r \Rightarrow A[k] \ge A[l]$$

```
[n > 1 ∧ H(A,1,n)]
r := 2;
while [γ1: 2 ≤ r ≤ n ∧ S(A,r) ∧ H(A,r,n) ∧ Max(A,r−1,n,r−1) ]
  r < n do [ decr n − r wrt > in Nat ]
    (if 2*r < n then t := 2*r else t := n;
     m := r;
     x := A[r];
     k := r+1;
     while [γ2: γ1 ∧ (t = 2*r ≤ n ∨ t = n ≤ 2*r) ∧ r < k ≤ t ∧ Max(A,r,k,m) ∧ x = A[m] ]
        k < t do [ decr t − k wrt > in Nat ]
          (if A[k] > x then x := A[k]; m := k else skip;
           k := k+1);
[α1: γ1 ∧ (t = 2*r ≤ n ∨ t = n ≤ 2*r) ∧ Max(A,r,t,m) ]
     if m > r then
       (A := swap(A,r,m);
        r := r+1;
        j := m;
        while [γ3: 2 < r ≤ n ∧ S(A,r) ∧ Hbut(A,r,n,j) ∧ Max(A,r−1,n,r−1) ]
           2*j < n do [ decr n − j wrt > in Nat ]
            (k := 2*j;
[α2: γ3 ∧ k = 2*j < n ]
             if k+1 < n then
               if A[j] ≥ A[k] and A[j] ≥ A[k+1] then j := n
               else if A[k] ≥ A[k+1]
                    then (A := swap(A,j,k);
[α3: 2 < r ≤ j ∧ k = 2*j < n−1 ∧ Hbut(A,r,n,k) ∧ S(A,r) ∧ Max(A,r−1,n,r−1) ]
                          j := k)
                    else (A := swap(A,j,k+1);
                          j := k+1)
             else if A[j] ≥ A[k] then j := n
                  else (A := swap(A,j,k);
                        j := k)
            )
       ) else r := r+1
  )
[S(A,n)]
```