Semantyka i weryfikacja programów 2024/25.
Egzamin 28/01/2025, zadanie 2 (weryfikacja)

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

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

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

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

zdefiniowane następująco:

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

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 $e\le e'\wedge e\neq e'$) oraz $e\ge e'$ (wyrażalne jako $e'\le e$). Dla czytelności, w formułach ciągi nierówności będziemy zapisywać w skróconej postaci, np. $j\le i\wedge i<k$ zapisując jako $j\le i<k$.

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

$$\begin{aligned}
&H(A,p)\equiv A{:}\mathit{Array}\wedge\forall l.1<l<p\Rightarrow A[l\ \texttt{div2}]\ge A[l]\\
&S(A,p,r)\equiv A{:}\mathit{Array}\wedge\forall l,l'.p\le l<l'<r\Rightarrow A[l]\le A[l']
\end{aligned}$$

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 γ_1 i γ_2 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,n)]
m := n;
[                                                                    ]
while [γ1:                                                           ]
  m > 2 do [ decr                wrt        in        ]
    (m := m-1;
[                                                                    ]
    A := swap(A,1,m);
[                                                                    ]
    j := 1;
[                                                                    ]
    while [γ2:                                                       ]
       2*j < m do [ decr             wrt        in        ]
        (k := 2*j;
[α1:                                                                 ]
        if k+1 < m
        then
          if A[j] ≥ A[k] and A[j] ≥ A[k+1] then j := m
          else if A[k] ≥ A[k+1]
               then (A := swap(A,j,k);
[                                                                    ]
                     j := k)
               else (A := swap(A,j,k+1);
[α2:                                                                 ]
                     j := k+1)
        else if A[j] ≥ A[k] then j := m
             else (A := swap(A,j,k);
[                                                                    ]
                   j := k)
        )
  );
[                                                                    ]
[S(A,1,n)]
```

Przyda się dodatkowy pomocniczy predykat:

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

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