Semantyka i weryfikacja programów 2021/22.
Egzamin 4/02/2022, zadanie 3 (weryfikacja)

Pracujemy w języku $\textsc{Tiny}_{\mathcal{A}}$ nad typem danych rozszerzonym 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 $i, j, n \in \mathbf{Int}$ 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 $e \le e' \wedge \neg(e = e')$) oraz $e > e'$ (wyrażalne jako $e' < e$). Dla czytelności, w formułach ciągi nierówności będziemy zapisywać w skróconej postaci, np. $j \le i \wedge i < k$ zapisując jako $j \le i < k$.

Dla wygody specyfikacji i dowodzenia poprawności wprowadzamy następujące schematy formuł:

$$\begin{aligned}
&\mathit{sorted}(A, j, k) \equiv A{:}\mathit{Array} \wedge \forall i.(j \le i < k \implies A[i] \le A[i+1])\\
&\mathit{leq}(A, j_1, k_1, j_2, k_2) \equiv\\
&\qquad A{:}\mathit{Array} \wedge \forall i_1, i_2.((j_1 \le i_1 \le k_1 \wedge j_2 \le i_2 \le k_2) \implies A[i_1] \le A[i_2])\\
&\mathit{minmax}(d, g, A, j, k) \equiv A{:}\mathit{Array} \wedge j \le d \le k \wedge j \le g \le k \wedge\\
&\qquad\qquad \forall i.(j \le i \le k \implies A[d] \le A[i] \le A[g])
\end{aligned}$$

Należy udowodnić całkowitą poprawność następującego programu względem podanych warunków:

$[A{:}Array \wedge n > 0\ ]$

```
 l := 0; p := n;
while l < p
   do (
        d := p; g := p
        i := p;
        while l < i
           do (
                i := i-1;
                if A[i] < A[d] then d := i
                else if A[i] > A[g] then g := i else skip;
              );
       A := swap(A,l,d);
       if g = l then g := d else skip;
       A := swap(A,g,p);
       l := l+1;
       if l < p then p := p-1 else skip
      )
```

$[A{:}Array \wedge sorted(A, 0, n)]$

Na następnej stronie podana jest wersja tego programu z miejscem na wpisanie odpowiednich anotacji, którą można wykorzystać dla przedstawienia rozwiązania (źródło LaTeXowe podane jest w osobnym pliku).

Wymagane jest:

- podanie niezmienników γ_1 i γ_2 obu pętli programu oraz asercji $\alpha_1, \alpha_2, \alpha_3, \alpha_4$, które „kodują" dowód poprawności (podanie pozostałych asercji jest opcjonalne, ale jeśli zostaną podane, to ewentualne błędy mogą wpłynąć na ocenę rozwiązania), oraz
- podanie anotacji [**decr ... in ... wrt ...**] tak, aby (w kontekście podanych niezmienników) wynikała z nich własność stopu pętli.

```
[A:Array ∧ n > 0 ]
l := 0; p := n;
[                                                                    ]
while [γ1 :                                                          ]
    l < p
    do [decr          in          wrt         ]
       ([                                                            ]
        d := p; g := p
        [α1 :                                                        ]
        i := p;
        [                                                            ]
        while [γ2 :                                                  ]
            l < i
            do [decr           in           wrt         ]
               ([                                                    ]
                i := i-1;
                [α2 :                                                ]
                if A[i] < A[d] then d := i
                else if A[i] > A[g] then g := i else skip
                [                                                    ]
               );
        [                                                            ]
        A := swap(A,l,d);
        [α3 :                                                        ]
        if g = l then g := d else skip;
        [                                                            ]
        A := swap(A,g,p);
        [                                                            ]
        l := l+1;
        [α4 :                                                        ]
        if l < p then p := p-1 else skip
        [                                                            ]
       )
[A:Array ∧ sorted(A, 0, n)]
```