Semantyka i weryfikacja programów 2021/22.
Egzamin II termin 22/02/2022, 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 $e \leq e' \wedge \neg(e = e')$) oraz $e > e'$ (wyrażalne jako $e' < e$). Dla czytelności, w formułach ciągi nierówności będziemy zapisywać w skróconej postaci, np. $j \leq i \wedge i < k$ zapisując jako $j \leq i < k$. Ponadto, konstrukcja `if` B `then` S (dla dowolnego wyrażenia logicznego B i instrukcji S) jak zwykle oznacza `if` B `then` S `else skip`.

Dla wygody specyfikacji i dowodzenia poprawności wprowadzamy następujące schematy formuł:

$$\begin{aligned}
&\mathit{Dominates}^{-}(A, j_1, k_1, j_2, k_2) \equiv A{:}\mathit{Array} \wedge \forall i_2.(j_2 \leq i_2 \leq k_2 \implies \exists i_1.(j_1 \leq i_1 \leq k_1 \wedge A[i_1] \leq A[i_2]))\\
&\mathit{MinFirst}(A, j, k) \equiv A{:}\mathit{Array} \wedge \forall i.(j \leq i \leq k \implies A[j] \leq A[i])
\end{aligned}$$

Należy udowodnić całkowitą poprawność następującego programu względem podanych warunków:

```
[ A:Array ∧ n ≥ 0 ]
d := n;
while
    d > 0
 do
   (
    k := 0;
    while
        2*k < d
      do
        (
         if A[k] > A[d-k] then A := swap(A,k,d-k);
         k := k+1
        );
    if 2*k = d then (if A[k] < A[k-1] then A := swap(A,k-1,k) );
    d := k-1;
   )
[ n ≥ 0 ∧ MinFirst(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 każdej z pętli programu oraz asercji α_1, α_2 i α_3, 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 ]
d := n;
[                                                                    ]
while [γ1 :                                                          ]
    d > 0
  do [decr            in      wrt    ]
    ([                                                               ]
     k := 0;
     [                                                               ]
     while [γ2 :                                                     ]
         2*k < d
       do [decr              in      wrt     ]
         ([                                                          ]
          if A[k] > A[d-k] then A := swap(A,k,d-k);
          [α1 :                                                      ]
          k := k+1
          [                                                          ]
         );
     [α2 :                                                           ]
     if 2*k = d then (if A[k] < A[k-1] then A := swap(A,k-1,k) );
     [α3 :                                                           ]
     d := k-1;
     [                                                               ]
    )
[                                                                    ]
[ n ≥ 0 ∧ MinFirst(A, 0, n) ]
```