Niezawodne Systemy Informatyczne

Lista 3

PRZEMYSŁAW KOBYLAŃSKI

Rozwiąż samodzielnie poniższe zadania i przedstaw prowadzącemu najpóźniej na 13. zajęciach.

Przy każdym z zadań podano maksymalną do uzyskania liczbę punktów.

Za zadanie otrzymuje się:
- 100% punktów gdy program **gnatprove** udowodnił wszystkie asercje (*Assert*), kontrakty (*Post*), kontrole indeksów (*index check*) i zakresów (*overflow check*) oraz nie użyto założeń (*Assume*);
- 75% punktów gdy program **gnatprove** udowodnił wszystkie asercje, kontrakty, kontrole indeksów;
- 50% punktów gdy program **gnatprove** udowodnił wszystkie kontrakty i kontrole indeksów (nieudowodniona asercja *Assert* będzie uznawana za użycie założenia *Assume* i wymaga uzasadnienia słownego podczas zaliczania).

Każde użyte założenie *Assume* należy udowodnić prowadzącemu podczas zaliczania listy ale zasadność jego użycia może być zakwestionowana przez prowadzącego a ma on głos rozstrzygający.

Wskazówka: poczytaj o parametrach polecenia **gnatprove** (*level*, *steps*, *timeout* i innych).

Zadanie 1 (16 pkt)

Poniżej przedstawiono kod procesora 6502[1] wyliczający iloczyn dwóch ośmiobitowych wartości A i B umieszczając szesnastobitowy wynik w miejscu AB:

```
init    lda  #$00
        sta  AB
        sta  AB+1
        sta  AA+1
        lda  A
        sta  AA
        lda  B
        beq  cont

loop    and  #$01
        beq  shift
        lda  AA
        clc
        adc  AB
        sta  AB
        lda  AA+1
        adc  AB+1
        sta  AB+1
shift   asl  AA
```

[1] O instrukcjach procesora 6502 możesz poczytać na stronie https://www.masswerk.at/6502/6502_instruction_set.html

```
        rol   AA+1
        lda   B
        lsr
        sta   B
        bne   loop
cont
```

Poniższa funkcja **Mult** w podobny sposób wylicza iloczyn liczb naturalnych **A** i **B** (**B1** jest lokalną kopią parametru **B**, który będąc w trybie **in** nie może zmieniać swojej wartości).

```
function Mult (A : Natural; B : Natural) return Natural
  with
  SPARK_Mode,
  Pre => A < 32768 and B < 32768,
  Post => Mult'Result = A * B
is
   AB : Natural := 0;
   AA : Natural := A;
   B1 : Natural := B;
begin
   while B1 > 0 loop
     if B1 mod 2 = 1 then
         AB := AB + AA;
      end if;
      AA := 2 * AA;
      B1 := B1 / 2;
   end loop;
   return AB;
end Mult;
```

Dopisz w funkcji **Mult** odpowiednie niezmienniki pętli aby program **gnatprove** udowodnił jej poprawność.

Napisz procedurę **Main**, która testuje poprawność działania funkcji **Mult**.

Zadanie 2 (8 pkt)

Poniżej przedstawiono fragment procedury **Rev**, która odwraca łańcuch znaków będący jej argumentem:

```
procedure Rev (S : in out String)
  with
  SPARK_Mode,
  Pre => S'First < Positive'Last / 2 and S'Last < Positive'Last / 2,
  Post => (for all I in S'Range => S(I) = S'Old(S'First + S'Last - I))
is
  …
begin
  …
end Rev;
```

Napisz procedurę **Rev** tak aby program **gnatprove** udowodnił, że procedura **Rev** poprawnie odwraca łańcuch znaków.

Napisz procedurę **Main**, która testuje poprawność działania procedury **Rev**.

Zadanie 3 (4 pkt)

Poniżej przedstawiono specyfikację pakietu **Bubble**, który deklaruje typ tablicowy **Arr** i procedurę **Sort**, sortującą tablice typu **Arr** przy użyciu algorytmu sortowania bąbelkowego.

Dodatkowo zdefiniowane są w nim dwie pomocnicze funkcje logiczne, które można wykorzystać w kontraktach i niezmiennikach:

- **Sorted (A)** — sprawdzająca czy tablica A jest posortowana,
- **Bubbled (A)** — sprawdzająca czy w tablicy A największa wartość znajduje się na ostatniej pozycji.

```
package Bubble with SPARK_Mode is

   type Arr is array (Integer range <>) of Integer;

   function Sorted (A : Arr) return Boolean is
      (for all I in A'First .. A'Last - 1 =>
               A(I) <= A(I + 1))
     with
       Ghost,
       Pre => A'Last > Integer'First,
       Post => Sorted'Result = (for all I in A'First .. A'Last - 1 =>
                                   A(I) <= A(I + 1));

   function Bubbled (A : Arr) return Boolean is
      (for all I in A'First .. A'Last - 1 =>
               A(I) <= A(A'Last))
     with
       Ghost,
       Pre => A'Last > Integer'First,
       Post => Bubbled'Result = (for all I in A'First .. A'Last - 1 =>
                                    A(I) <= A(A'Last));

   procedure Sort (A : in out Arr)
     with
     Pre => A'Last > Integer'First and A'Last < Integer'Last,
     Post =>
       Sorted (A) and
       (for all I in A'Range =>
                (for some J in A'Range =>
                         A(I) = A'Old(J))) and
       (for all I in A'Range =>
                (for some J in A'Range =>
                         A'Old(I) = A(J)));

end Bubble;
```

Zwróć uwagę na dwa ostatnie warunki końcowe w kontrakcie procedury **Sort**. Pierwszy z nich gwarantuje, że w wyjściowej tablicy nie pojawi się żadna wartość, której nie było w tablicy wejściowej. Drugi, natomiast, gwarantuje, że wszystkie wartości jakie były w tablicy wejściowej znajdą się w tablicy wyjściowej (nie uwzględniają jednak one krotności wystąpienia wartości np. (2, 1, 2) —> (1, 1, 2)).

Napisz treść pakietu **Bubble** tak aby program **gnatprove** udowodnił poprawność działania procedury **Sort**.

Napisz procedurę **Main**, która testuje poprawność działania procedury **Sort**.

Wskazówka

Jeśli masz problemy z udowodnieniem poprawności procedury **Sort** nawet na czwartym poziomie (parametr --**level=4**), to utwórz trzy wersje pakietu:

- W pierwszej pozostaw pierwszy warunek końcowy i tylko te niezmienniki, które są potrzebne do jego udowodnienia.
- W drugiej pozostaw drugi warunek końcowy i tylko te niezmienniki, które są potrzebne do jego udowodnienia.
- W trzeciej pozostaw trzeci warunek końcowy i tylko te niezmienniki, które są potrzebne do jego udowodnienia.

Poprawność każdej wersji pakietu udowodnij osobno programem **gnatprove**.