Niezawodne Systemy Informatyczne

Przemysław Kobylański
Katedra Podstaw Informatyki (K68W11D03)
Politechnika Wrocławska

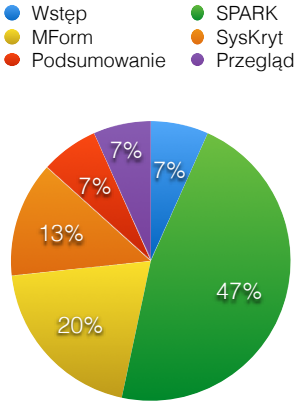
Na podstawie

J.W. McCormick, P.C. Chapin. *Building High Integrity Applications with SPARK*
K.R. Apt, F.S. de Boer, E-R Olderog *Verification of Sequential and Concurrent Programs*
I. Sommerville *Inżynieria oprogramowania*

Politechnika Wrocławska

Plan wykładu

- Wstęp (2h)
- SPARK (14h)
- Metody formalne (6h)
- Systemy krytyczne (4h)
- Przegląd innych narzędzi (2h)
- Podsumowanie (2h)



Literatura podstawowa

- J.W. McCormick, P.C. Chapin. **Building High Integrity Applications with SPARK**. Cambridge University Press, 2015.
- K.R. Apt, F.S. de Boer, E-R Olderog. **Verification of Sequential and Concurrent Programs**. Springer, 2009.
- I. Sommerville. **Inżynieria oprogramowania**. WNT, Warszawa, 2003.

Narzędzia

- **SPARK 2014**
  - **GNATprove** wersja z roku 2020
    - **Why3** wersja 1.2.1
      - **Alt-ergo** wersja 2.3.0
      - **CVC4** wersja 1.8
      - **Z3** wersja 4.8.6

Zasady zaliczenia

- Ocena zależy od jest średniej ważonej uzyskanych punktów z laboratorium (0-100 z wagą 40%) i kartkówek (0-100 z wagą 60%).
- Aby zaliczyć grupę kursów należy z laboratorium otrzymać co najmniej 50 punktów i z kartkówek co najmniej 50 punktów.
- Listy zadań na laboratorium należy rozwiązywać samodzielnie.
- Kartkówki odbędą się na ostatnich wykładach.
- Więcej na stronie WWW kursu (między innymi prezentacja, listy zadań, terminy kartkówek).

Wyliczanie oceny końcowej

```
Pre =>
 LAB >= 50.0 and LAB <= 100.0 and
 KOL >= 50.0 and KOL <= 100.0,

Contract_Cases =>
 (
  0.4*LAB + 0.6*KOL <  50.0 => ZAL = 2.0,
  0.4*LAB + 0.6*KOL <  60.0 => ZAL = 3.0,
  0.4*LAB + 0.6*KOL <  70.0 => ZAL = 3.5,
  0.4*LAB + 0.6*KOL <  80.0 => ZAL = 4.0,
  0.4*LAB + 0.6*KOL <  90.0 => ZAL = 4.5,
  others                    => ZAL = 5.0)
```

Przykład

Poszukiwanie największej wartości w tablicy.

find.ads **specyfikacja**

```
-- wersja 1
package Find with SPARK_Mode is
    type Vector is array (Integer range <>) of Integer;
    function Max (V : Vector) return Integer
    with
        Post => (for all I in V'Range => V(I) <= Max'Result);
end Find;
```

find.adb **treść**

```
package body Find with SPARK_Mode is
    function Max (V : Vector) return Integer is
        M : Integer := Integer'Last;
    begin
        return M;
    end Max;
end Find;
```

Przykład cd.

```
-- wersja 2
package Find with SPARK_Mode is
    type Vector is array (Integer range <>) of Integer;
    function Max (V : Vector) return Integer
    with
        Post => (for all I in V'Range => V(I) <= Max'Result) and
                (for some I in V'Range => V(I) = Max'Result);
end Find;

package body Find with SPARK_Mode is
    function Max (V : Vector) return Integer is
        M : Integer := Integer'Last;
    begin
        return M;
    end Max;
end Find;
```

gnatprove: postcondition might fail (e.g. V'First = 1 and V'Last = 0)

Przykład cd.

```
-- wersja 3
package Find with SPARK_Mode is
    type Vector is array (Integer range <>) of Integer;
    function Max (V : Vector) return Integer
    with
        Pre => V'Length > 0,
        Post => (for all I in V'Range => V(I) <= Max'Result) and
                (for some I in V'Range => V(I) = Max'Result);
end Find;

package body Find with SPARK_Mode is
    function Max (V : Vector) return Integer is
        M : Integer := Integer'Last;
    begin
        return M;
    end Max;
end Find;
```

gnatprove: postcondition might fail (e.g. Max'Result = 2147483647 and V = (others => 0) and V'First = 0 and V'Last = 0)

Przykład cd.

```
-- wersja 4
package Find with SPARK_Mode is
    type Vector is array (Integer range <>) of Integer;
    function Max (V : Vector) return Integer
    with
        Pre => V'Length > 0,
        Post => (for all I in V'Range => V(I) <= Max'Result) and
                (for some I in V'Range => V(I) = Max'Result);
end Find;

package body Find with SPARK_Mode is
    function Max (V : Vector) return Integer is
        M : Integer := V(V'First);
    begin
        for I in V'First + 1 .. V'Last loop
            if V(I) > M then
              M := V(I);
            end if;
        end loop;
        return M;
    end Max;
end Find;
```

gnatprove: overflow check might fail (e.g. V'First = 2147483647)

Przykład cd.

```
-- wersja 5
package Find with SPARK_Mode is
    type Vector is array (Integer range <>) of Integer;
    function Max (V : Vector) return Integer
    with
        Pre => V'Length > 0 and
               V'First < Integer'Last,
        Post => (for all I in V'Range => V(I) <= Max'Result) and
                (for some I in V'Range => V(I) = Max'Result);
end Find;

package body Find with SPARK_Mode is
    function Max (V : Vector) return Integer is
        M : Integer := V(V'First);
    begin
        for I in V'First + 1 .. V'Last loop
            if V(I) > M then
              M := V(I);
            end if;
        end loop;
        return M;
    end Max;
end Find;
```

gnatprove: postcondition might fail
cannot prove V(I) <= Max'Result

Przykład cd.

```
-- wersja 6
package Find with SPARK_Mode is
    type Vector is array (Integer range <>) of Integer;
    function Max (V : Vector) return Integer
    with
        Pre => V'Length > 0 and
               V'First < Integer'Last,
        Post => (for all I in V'Range => V(I) <= Max'Result) and
                (for some I in V'Range => V(I) = Max'Result);
end Find;

package body Find with SPARK_Mode is
    function Max (V : Vector) return Integer is
        M : Integer := V(V'First);
    begin
        for I in V'First + 1 .. V'Last loop
            pragma Loop_Invariant (for all J in V'First .. I - 1 =>
                                     V(J) <= M);
            pragma Loop_Invariant (for some J in V'First .. I - 1 =>
                                     V(J) = M);
            if V(I) > M then
              M := V(I);
            end if;
        end loop;
        return M;
    end Max;
end Find;
```

Przykład cd.

Prosty test:

main.adb

```
with Ada.Integer_Text_IO;
with Find;
procedure Main is
begin
   Ada.Integer_Text_IO.Put (Find.Max ((3, 1, 5, 2, 7, 4, 3, 1)));
end Main;
```

Opis projektu:

find.gpr

```
project Find is
    for Source_Dirs use ("src");
    for Object_Dir use "obj";
    for Main use ("main.adb");
end Find;
```

Przykład cd.

Makefile

```
all:
        gprbuild -p

prove:
        gnatprove -P find.gpr -j0

clean:
        gprclean
        gnatprove -P find.gpr --clean
```

kompilacja

weryfikacja

sprzątanie

Przykład cd.

```
$ make prove
gnatprove -P find.gpr -j0
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
Summary logged in obj/gnatprove/gnatprove.out
$ cat obj/gnatprove/gnatprove.out
Summary of SPARK analysis
=========================

-----------------------------------------------------------------------------------------------------
SPARK Analysis results        Total       Flow    Interval   CodePeer      Provers   Justified   Unproved
-----------------------------------------------------------------------------------------------------
Data Dependencies                 .          .          .          .            .           .          .
Flow Dependencies                 .          .          .          .            .           .          .
Initialization                    4          4          .          .            .           .          .
Non-Aliasing                      .          .          .          .            .           .          .
Run-time Checks                  10          .          .          .    10 (CVC4)           .          .
Assertions                        4          .          .          .     4 (CVC4)           .          .
Functional Contracts              1          .          .          .     1 (CVC4)           .          .
LSP Verification                  .          .          .          .            .           .          .
-----------------------------------------------------------------------------------------------------
Total                            19     4 (21%)         .          .    15 (79%)           .          .


Analyzed 2 units
in unit main, 0 subprograms and packages out of 1 analyzed
  Main at main.adb:4 skipped
in unit find, 2 subprograms and packages out of 2 analyzed
  Find at find.ads:1 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (0 checks)
  Find.Max at find.ads:5 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (15 checks)
```

SPARK

- Elementy języka SPARK (6h)
- Pakiety (3h)
- Sprawdzanie zależności (2h)
- Dowodzenie (3h)