1 Podstawy stosowania pakietu listings

W trywialnym przypadku program z języka C++ może składać się tylko z jednego pliku i jednej funkcji. *Funkcja* jest wydzieloną częścią programu, realizującą pewne zadanie. Kompletny program musi zawierać funkcję o nazwie *main* od której rozpoczyna się wykonanie programu. Do programu można dołączać pliki zawierające nagłówki (opis) funkcji zdefiniowanych w innych plikach lub funkcji systemowych (dyrektywa *include*).

```
1 #include <iostream>
2
3 int main()
4 {
5   std::cout << "C++\n";
6 }
```

Komentarz w C++, to dowolnej długości tekst ograniczony znakami `/*` i `*/` lub tekst od znaku `//` do końca linii.

```
int main() {  /* Ten program nic nie robi. */ }
```

Ada należy do języków rodziny Algol/Pascal, programy napisane w Adzie są czytelne i stosunkowo łatwe do analizy.

```
with Text_IO, Ada.Integer_Text_IO;                1
use  Text_IO, Ada.Integer_Text_IO;                2
                                                  3
procedure Silnia is                               4
  n, s, i : Integer := 1;                         5
                                                  6
begin                                             7
  Get(n);                                         8
                                                  9
  while i < n loop                                10
    i := i + 1;                                   11
    s := s * i;                                   12
  end loop;                                       13
                                                  14
  Put("Silnia: ");                                15
  Put(s, 0);                                      16
end;                                              17
```

2 Definiowanie własnego języka programowania

Język opisu dynamiki (*Alvis Code Language*, w skrócie AlvisCD, służy do definiowania zachowania indywidualnych agentów. Występujące w nim instrukcje są wzorowane na wybranych operatorach algebry CCS, które w XCCS stosowane były w warstwie algebraicznej. W przeciwieństwie do języków modelowania CCS i XCCS, które skupiają się głównie na opisie komunikacji, marginalizując kwestie związane z manipulowaniem wartościami parametrów, język Alvis pozwala na wygodne modyfikacje wartości parametrów agenta, niezależne od instrukcji dotyczących komunikacji.

```
1  environment {
2    in wakeup [] (map (60000*) [1..]) durable;
3    in off [] (map (1000*) [1..]) signal;
4    out warning [0,1,2] [];
5    out brake [] [];
6  }
7
8  agent ATS {
9    loop {                              -- 1
10     in wakeup;                        -- 2
11     out warning 1;                    -- 3
12     select {                          -- 4
13       alt (ready [in(off)]) {
14         in off;                       -- 5
15         out warning 0;                -- 6
16       }
17       alt(delay 6000) {
18         out warning 2;                -- 7
19         select {                      -- 8
20           alt (ready [in(off)]) {
21             in off;                   -- 9
22             out warning 0;            -- 10
23           }
24           alt (delay 3000) {
25             out brake;                -- 11
26             exit;                     -- 12
27           }
28         }
29       }
30     }
31   }
32 }
```

UWAGA: Pakiet listings pamięta poprzednie ustawienia, jeśli ich nie nadpiszemy, tzn. jeżeli dla poprzedniej specyfikacji ustawiliśmy wyświetlanie numerów linii po prawej stronie, a w kolejnej specyfikacji nic o liczbach nie piszemy, to nadal będą wyświetlane po prawej stronie.

3 Podpisy, odwołania i ramki

```
1  agent Buffer {
2    i :: Int = 0;
3    proc pop  { out pop i; }
4    proc push { in  push i; }
5  }
```

Listing 1: Przykład agenta pasywnego

Agenty pasywne są stosowane do opisu współdzielonych zasobów. Przykładową implementację jednokomórkowego bufora pokazano na listingu 1.

```
12 agent Buffer {
13   i :: Int = 0;
14   proc pop  { out pop i; }
15   proc push { in  push i; }
16 }
```

Listing 2: Przykład agenta pasywnego

Na listingu 2 ciągle rozważamy ten sam przykład, ale w ramce ;)

Listing 3: Przykład agenta pasywnego

```
1  agent Buffer {
2    i :: Int = 0;
3    proc pop  { out pop i; }
4    proc push { in  push i; }
5  }
```

Listing 4: Przykład agenta pasywnego

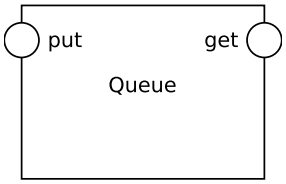
```
1  agent Buffer {
2    i :: Int = 0;
3    proc pop  { out pop i; }
4    proc push { in  push i; }
5  }
```

Listing 5: Przykład agenta pasywnego

```
agent Buffer {
  i :: Int = 0;
  proc pop  { out pop i; }
  proc push { in  push i; }
}
```

Narożniki podajemy od prawego górnego począwszy zgodnie z ruchem wskazówek zegara. Należy podać dokładnie 4 litery, *t* oznacza zaokrąglony narożnik, a *f* prosty.

Można połączyć grafikę z kodem w ramach jednego środowiska *figure*, tak jak pokazano na rysunku 1.




```
agent Queue {
  q :: [Int] = [];
  n :: Int = 0;

  proc put {
    critical {
      in put n;
      q = q ++ [n]; }
  }
  proc get (q /= []) {
    critical {
      n = head q;
      q = tail q;
      out get n; }
  }
}
```

Rysunek 1: Agent *Queue*