Teoria Współbieżności
Ćwiczenie 10

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z notacją Communicating Sequential Processes (w skrócie CSP) do opisu współbieżności i niektórych problemów algorytmicznych.

2 Przypomnienie

Poniżej sciąga z wybranych poleceń notacji CSP niezbędnych i wystarczających do rozwiązania poniższych zadań. Polecenia są na podstawie artykułu Hoare [1], dużo obszerniesze można podejście można znaleźć w [3].

Zmienne

Można zadeklarować zmienne, kilka zmiennych po przecinku lub tablicę.
Struktura: *nazwa_zmiennej : typ*
Struktura dla tablicy: *nazwa_zmiennej : (od .. do) typ*

```
check : boolean;
a, b : integer;
pi : real;
tab : (1..100) integer
```

Przypisanie wartości.
Struktura: *nazwa_zmiennej := wyrażenie*

```
a := b ;
data := serialize( tab(3) ) ;
(x,y) := (y,x)
```

Uwaga: ostatnie działa jak w pytonowskie x,y = y,x .

Złożenie sekwencyjne

Średnik działa jak and - łączy polecenia jak & w bashu. Zmienne istnieją w ramach bloków [*instrukcje*] .
Struktura: *instrukcja ; instrukcja ; ...*

```
[a, b: integer;
  a := 1; b := 2;
  [ tmp: integer;
    tmp := a; a := b; b := tmp   ] ]
```

If

Działa jak if then elseif elseif elseif... z taką mofyfikacją, że warunki są sprawdzne niedeterministycznie a któryś z warunków (dozór) musi się wykonać.
Struktura: [*dozór1* $->$ *instrukcje1* | *dozór2* $->$ *instrukcje2* | *...*]

```
y := foo(x);
z := false;
[ x < 0 -> y := -y ; z :=true
| x > 0 ; x = 0 -> skip]
```

Uwaga: powyżej przykład if then bez elsa dzieki skip - to pusta instrukcja jak pass w python. Wyrażenia można łączyć koniunkcją dzięki średnikowi ; .

Pętla

Pętla w działaniu przypomina whilea a budową ifa - z tym, że jest przerywana gdy dopiero wtedy gdy żaden dozór nie będzie prawdziwy. Jeśli kilka dozorów jest poprawnych nie wiadomo, który zostanie odpalony.
Struktura: *[*dozór1* $->$ *instrukcje1* | *dozór2* $->$ *instrukcje2* | *...*]

```
suma := 0; i := 1;
*[ i ⩽ 100 -> suma := suma + 1; i := i + 1 ]
```

Powyżej imitacja fora z dodawaniem stu jedynek.

```
*[ (i: 1..100) tab(i) ≠ 0  -> tab(i) := 0 ]
```

Uwaga: można zbiorczo zadeklarować wiele dozorów; powyżej przykład w zerowaniu tablicy.

Złożenie równoległe

Obliczenia można wykonywać równolegle.
Struktura: [*instrukcje1* || *instrukcje2* || *...*]

```
x := 1; y := 0;
 [  x := x + x || y :=  y mod 10 ]
```

Procesy (instrukcje wykonywane równolegle) można nazwać.
Struktura:[*nazwa_procesu1 :: instrukcje1* || *nazwa_procesu2 :: instrukcje2* || ...]

```
x := 1; y := 0;
 [ ProcesA:: x := x + x || ProcesB:: y := y mod 10 ]
```

Kanały

Procesy mogą się komunikować poprzez kanały. Przy wysyłaniu nadawca czeka na odbiór wiadomości, podobnie czeka odbiorca na nadawcę.
Wysyłanie: *nazwa_procesu ! zmienna*
Odbieranie: *nazwa_procesu ? zmienna*

```
x := 3;
 [ ProcA::
  msg : integer ;
  msg := x+x ;
  Q!msg
 || ProcB::
  msg : integer ;
  ProcA?msg ;
  x := msg ]
```

Uwaga: odbieranie wiadomości może być w dozorze (wysyłanie nie); pętla przestanie się wykonywać dopiero gdy proces P będzie zakończony (bo wtedy P?msg będzie fałszem) w innym przypadku czeka na nadanie wiadomości.

```
*[ P?msg -> data := foo(msg) ]
```

Tablica procesów

Można stwożyć tablicę procesów i użyć ich identyfikatora w ciele instrukcji. Do komunikacji można używać zapisu *nazwa_procesu* (*zmienna*) *? zmienna* lub *nazwa_procesu* (*zmienna*) *! zmienna*

```
[ ProcP(i: 1..10)::
  x , z : integer ;
  x:= hasz(i) ;
  ProcQ(i)!chiper(i,x)  ;
  ProcK(i)?z ]
```

3 Ćwiczenia

Niektóre z przedstawionych problemów są sekwencyjne. Jeśli uważasz, że polecenie nie jest wystarczająco sprecyzowane, możesz założyć swoją wersje zadania; na potrzeby zadań możesz użyć i nie definiować funkcje np. Produkuj() foo(a,b) Czytaj(z).
Napisz rozwiązanie w notacji w CSP, które:

1. nic nie robi
2. liczy średnią z tablicy liczb rzeczywistych
3. sprawdzi czy zadana wartość jest w tablicy
4. odwróci elementy w tablicy
5. sortuje tablicę
6. oblicza równolegle pierwiatki równania kwadratowego.
7. sprawdza czy liczba jest pierwsza
8. liczy silnię
9. liczy liczbę fibbonaciego
10. symuluje 1 producenta i 1 konsumenta bez buforu
11. symuluje 1 producenta i 1 konsumenta z buforem 1 elementowym
12. symuluje N producentów i 1 konsumenta z buforem
13. symuluje 1 producenta i wielu konsumentów
14. symuluje wielu producentów i wielu konsumentów wraz z buforem
15. symuluje problem 5 filozofów (dowolna wersja, każda punktowana)
16. symuluje problem czytelników i pisarzy (dowolna wersja, każda punktowana)

Literatura

[1] Hoare, Charles Antony Richard. ”Communicating sequential processes.” Communications of the ACM 21.8 (1978): 666-677.

[2] Engel, Marcin. Programowanie współbieżne i rozproszone, wazniak.mimuw.edu.pl

[3] Hoare, C. A. R. Communicating Sequential Processes, Prentice Hall, 1985

Uwagi: sluzalec@agh.edu.pl