

Teoria Współbieżności

Ćwiczenie 13

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 sציąga z wybranych poleceń notacji CSP niezbędnych i wystarczających do rozwiązania poniższych zadań. Polecenia są na podstawie artykułu Hoare [1], dużo obszerniejsze 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*

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

Przypisanie wartości.

Struktura: *nazwa_zmiennej* := *wyrażenie*

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

Uwaga: ostatnie działa jak w pythonowskie $x,y = y,x$.

Złożenie sekwencyjne

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

Struktura: *instrukcja ; instrukcja ; ...*

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

If

Działa jak if then elseif elseif... z taką modyfikacją, że warunki są sprawdzane niedeterministycznie a któryś z warunków (dozór) musi się wykonać.

Struktura: [*dozór1* -> *instrukcje1* | *dozór2* -> *instrukcje2* | ...]

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

Uwaga: powyżej przykład if then bez elsa dzięki 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* | ...]

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

Powyżej imitacja for'a z dodawaniem stu jedynek.

```
1 *[ (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 równoległe.

Struktura: [*instrukcje1* || *instrukcje2* || ...]

```

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

```

Procesy (instrukcje wykonywane równolegle) można nazwać.

Struktura: *[nazwa_procesu1 :: instrukcje1 || nazwa_procesu2 :: instrukcje2 || ...]*

```

1 x := 1; y := 0;
2 [ 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*

```

1 x := 3;
2 [ ProcA ::
3   msg : integer ;
4   msg := x+x ;
5   Q!msg
6 || ProcB ::
7   msg : integer ;
8   ProcA?msg ;
9   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.

```

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

```

Tablica procesów

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

```

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

```

3 Ćwiczenia

Niektóre z przedstawionych problemów są sekwencyjne. Jeśli uważasz, że polecenie nie jest wystarczająco sprecyzowane, możesz założyć swoją wersję 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. działa w nieskończoność (pętli się)
3. liczy średnią z tablicy liczb rzeczywistych
4. sprawdzi czy zadana wartość jest w tablicy
5. odwróci elementy w tablicy
6. sortuje tablicę (dowolne sortowanie, każde punktowane)
7. oblicza równoległe pierwiatki równania kwadratowego.
8. sprawdza czy liczba jest pierwsza
9. liczy silnię
10. liczy liczbę fibbonaciego
11. poda w dowolnej kolejności liczby parzyste od 1 do 1000
12. sprawdzi dla tablicy czy jakaś jej (ciągła) podtablica sumuje się do 0
13. symuluje producenta - konsumenta, na przykład wersje:
 - 13.1. 1 producenta i 1 konsumenta bez buforu
 - 13.2. 1 producenta i 1 konsumenta z buforem 1 elementowym
 - 13.3. N producentów i 1 konsumenta z buforem
 - 13.4. 1 producenta i wielu konsumentów
 - 13.5. wielu producentów i wielu konsumentów wraz z buforem
14. symuluje problem 5 filozofów (dowolna wersja, każda punktowana)
15. 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