CSP

Паралелна команда

- Паралелна команда служи за специфицирање процеса који се извршавају конкурентно и има следећи облик:

  [proc1::impl1 || ... || procN::implN]

Наредба доделе

`x:=x+1`	вредност *x* након доделе је за један већа од вредности *x* пре доделе
`(x,y):=(y,x)`	*x* и *y* размењују вредност
`x:=cons(left,right)`	креира се сложена вредност и додељује променљивој *x*
`cons(left,right):=x`	ако *x* има структуру облика *cons(a,b)* онда се врши додела *left:=a* и *right:=b*; у супротном долази до грешке
`insert(n):=insert(2*x+1)`	еквивалентно са *n:=2*x+1*
`c:=P()`	променљивој *c* се додељује вредност сигнала *P()*
`P():=c`	није дозвољено јер сигнал не може добити вредност
`insert(n):=has(n)`	није дозвољено јер су конструктори различити

Комуникационе примитиве

- Могућа је само синхрона *point-to-point* комуникација, уз директно именовање процеса. Примитиве су следеће:

  <process>!<message> – слање поруке <*message*> процесу <*process*>

  <process>?<message> – пријем поруке <*message*> од процеса <*process*>

Комуникационе примитиве

`X!m`	пошаљи вредност променљиве *m* (која може бити скалар, низ, итд.) процесу *X*
`X?m`	прими поруку од процеса *X* и смести је у променљиву *m*
`X!(3*a+b,13)`	пошаљи поруку процесу *X* у виду сложеног израза са два елемента
`X?(x,y)`	прими поруку од процеса *X* у виду сложеног израза са два елемента и додели одговарајуће вредности променљивама *x* и *y*
`console(j-1)!"A"`	пошаљи знак "A" *j-1*-вом у низу процеса *console*
`console(i)?c`	прими вредност од *i*-тог у низу процеса *console* и додели је променљивој *c*
`X(i)?V()`	прими сигнал *V()* од *i*-тог у низу процеса *X*, тј. чекај да процес *X* пошаље сигнал *V()* и дотле не примај друге поруке
`sem!P()`	пошаљи сигнал *P()* процесу *sem*

Контролне структуре

- Од основних контролних структура CSP има алтернативну и репетитивну алтернативну команду:
- Алтернативна команда има следећи облик:

  [guard1 → statement list1

  [] guard2 → statement list2

  ...

  [] guardn → statement listn

  ]
- Алтернативна команда има и скраћени облик:

  [(i:1..n) guard → statement list]

Контролне структуре

- Репетитивна алтернативна команда (често се назива и итеративна команда) има веома сличну синтаксу као и обична алтернативна команда, с том разликом да се алтернативна команда понавља све док се не деси да ни један услов није задовољен:

  *[guard1 → statement list1

  [] guard2 → statement list2

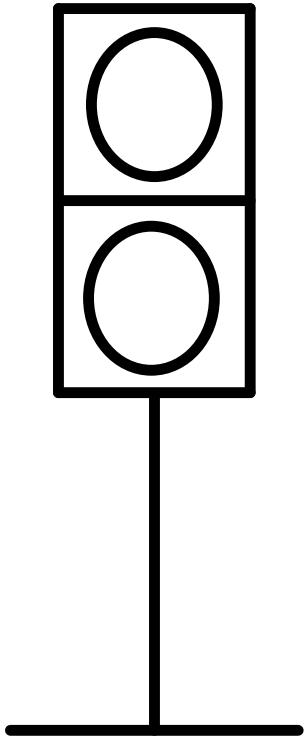
  ...

  [] guardn → statement listn

  ]

Задаци

Семафор



Семафор

Пројектовати бинарни семафор користећи програмски језик CSP.

Семафор

```
s : integer; s : = 0;
    *[
            s > 0; (i:1..100) X(i)?wait() -> s := s - 1
            []
            (i:1..100) X(i)?signal() -> s : = s + 1
    ]
```

Прослеђивање текста

Написати програм за процес X који прослеђује знаке добијене од процеса *west* процесу *east*.

Прослеђивање текста

X::*[c: character; west?c -> east!c]

Прослеђивање текста

Модификовати претходно решење тако да свака две суседне звездице “**” замени са “^”. Сматрати да последњи знак није звездица.

Прослеђивање текста

```
X::* [      c: character;
            west?c ->
                        [           c <> '*' ->east!c;
                                    []
                                    c = '*' -> west?c;
                                    [           c <> '*' -> east!'*'; east!c;
                                                []
                                                c = '*' -> east!'^'
                                    ]
                        ]
    ]
```

Producer - Consumer

Producer - Consumer

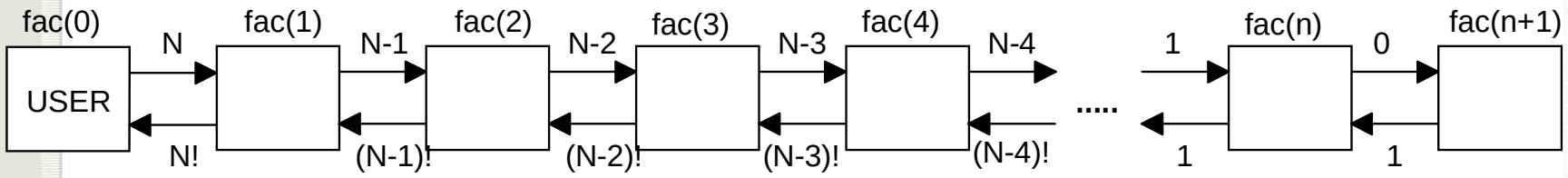
```
[Producer (i:1..5)::PRODUCER || Consumer (i:1..6)::CONSUMER
|| Buffer::BUFFER]
BUFFER:: [buffer: (0..9) portion;
    in, out : integer; in:=0; out:=0;
    *[
        in<out + 10; (i: 1..5) Producer (i)?buffer(in mod 10) ->
                in:=in + 1
        []
        out < in; (i: 1..6) Consumer (i)?more() ->
                Consumer (i)!buffer(out mod 10) -> out:=out + 1
    ]
]
```

Producer - Consumer

```
PRODUCER:: *[ data: portion;
    PRODUCE;
    Buffer!data
]
CONSUMER:: *[ data: portion;
    Buffer!more();
    Buffer?data;
    CONSUME
]
```

Факторијел

Израчунати факторијел рекурзивном методом, до одређене границе.



Факторијел

```
[    fac (I: 1..LIMIT):
     *[N: integer; fac (i-1)?n ->
            [           n = 0 ->  fac (i-1)!1
                        []
                        n > 0 ->  fac (i + 1)!n-1;
                                  r: integer; fac(i + 1)?r;
                                  fac (i - 1)!(n*r)
            ]
     ]

     || fac (0):: USER
]
```

У USER-? има fac(1)!n.

Скуп

- Представити у CSP-у скуп од највише 100 целих бројева као процес S, који прихвата два типа инструкција од позивајућег процеса X:

1) S!*insert (n)*, убацује цео број n у скуп, и

2) S!*has (n);...;S?b*, где је b истинито ако је n у скупу, односно неистинито у супротном случају.

У почетку је скуп празан.

Скуп

```
S:: content: (0..99) integer;
    size: integer; size := 0;
    *[      n: integer;
            X?has (n) ->SEARCH; X!(i < size)
    []
            n: integer;
            X?insert (n) -> SEARCH;
                    [           i < size -> skip
                                []
                                i = size; size < 100 ->
                                            content (size) := n; size := size + 1
                    ]
    ]
SEARCH:
i: integer; i := 0;
*[i < size; content (i) <> n -> i := i + 1
```

Скуп

- Проширити претходно решење, обезбеђујући брзи метод за сканирање свих елемената скупа, без мењања вредности. Кориснички програм садржи команду типа:

S!scan (); more: **boolean**; more := true;

*[more; x: **integer**; S?next (x); -> ...radi sa x...

[]

more; S?noneleft () -> more := false

]

где S!*scan ()* служи да постави скуп у сканирајући режим.

Скуп

Додаћемо трећу заштићену команду у спољну петљу претходног решења:

```
[]
X?scan () ->        i: integer; i := 0;
            *[i < size ->          X!next (content (i));
                                   i := i + 1;
            ]
            X!noneleft ()
```

Скуп

- Решити проблем скупа од максимално 100 бројева, са две операције (из претпоследњег задатка) помоћу низа процеса, од којих сваки садржи највише један број. Када процес не садржи ниједан број, на сваки упит о садржини треба да одговори са "false".

Скуп

Процес се налази у почетном стању када нема садржај. По првом убацивању елемента у тај процес, он мења стање у коме сада прима поруке од претходног процеса и, евентуално, шаље податак следећем.

Позивајући процес је S(0) који, када хоће да пошаље податак у скуп, шаље га ка процесу S(1), а када испитује да ли је податак у скупу ради следеће:

S(1)!has (n);...;[(i: 1..100) S(i)?b -> **skip**]

Због ефикасности, скуп треба да буде сортиран.

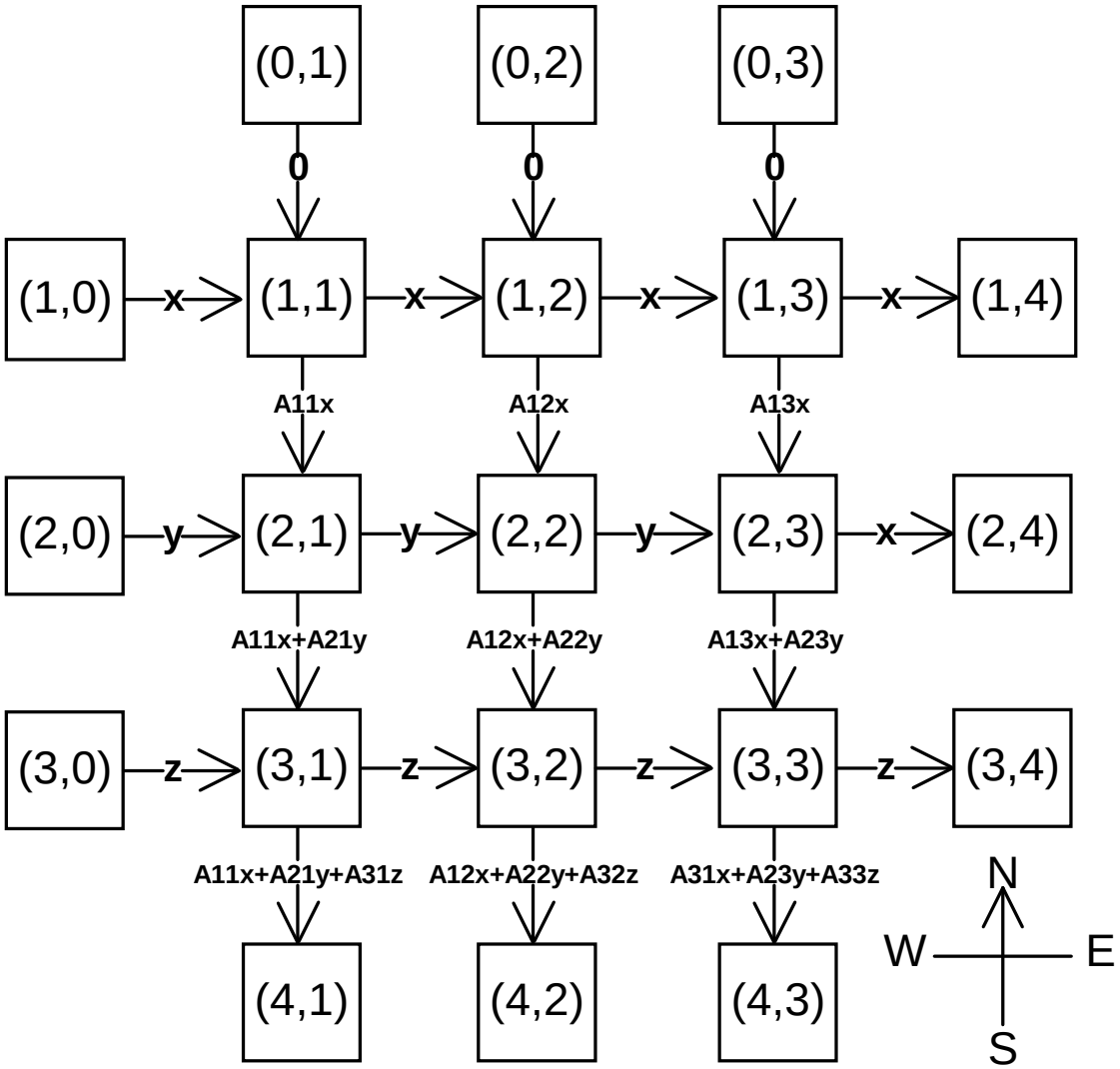
Скуп

```
S (i:1..100):
*[   n: integer; S (i-1)?has (n) -> S(0)!false
 []
     n: integer; S (i-1)?insert (n) ->
            *[ m: integer; S (i-1)?has (m)
                        [           m <= n  -> S(0)!(m = n)
                        []
                                    m > n  -> S(i + 1)!has (m)
                        ]
            []
                        m: integer; S (i - 1)?insert (m)
                        [           m < n  -> S (i + 1)!insert (n); n := m
                        []
                                    m = n  -> skip
                        []
                                    m > n  -> S (i + 1)!insert (m)
                        ]
            ]
 ]
```

Множење матрице

Дата је квадратна матрица A реда 3. На улаз долазе три низа података, од којих сваки представља по један елемент вектора IN. Три низа података треба да се појаве на излазу, од којих сваки представља по један елемент производа INA. После почетног кашњења, резултати треба да се производе истом брзином којом стиже улаз. Матрица A је фиксна.

Множење матрице



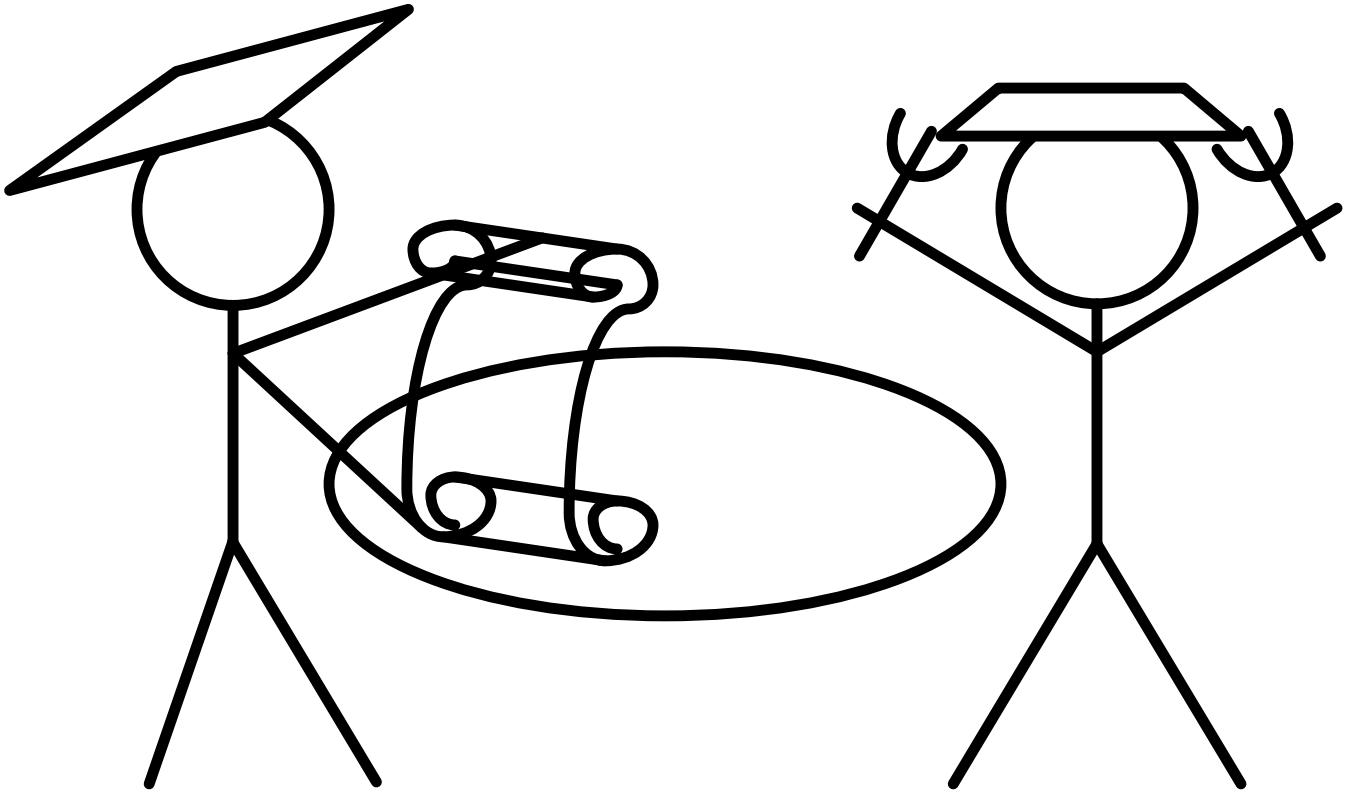
Множење матрице

```
[M (i:1..3, 0)::WEST || M (0, j:1..3)::NORTH || M (i:1..3, 4)::EAST ||
    M (4, j:1..3)::SOUTH || M (i:1..3, j:1..3)::CENTER]
```

Процеси WEST и SOUTH су кориснички програми. а преостали су:

```
NORTH:: *[true -> M (1, j)!0]
EAST::   *[x: real; M (i, 3)?x -> skip]
CENTER:: *[x: real; M (i, j-1)?x ->
                        M (i, j + 1)!x
                        sum: real;
                        M (i - 1, j)?sum;
                        M (i + 1, j)!(A (i, j)*x + sum)
          ]
```

Dining philosophers problem



Dining philosophers problem

```
[Pfil (i:0..4) :: PFILOSOPHER || Fork (i:0..4) :: FORK || Room :: ROOM]

PFILOSOPHER :: [left, right : integer; left := i; right := (i +1) mod 5;
    *[true -> THINK;
                                    room!ticket();
                                    fork(rihgt)!in();
                                    fork(left)!in();
                                    EAT;
                                    fork(left)!out();
                                    fork(right)!out();
                                    room!back();
    ]
]
```

Dining philosophers problem

```
FORK:: [left, right : integer; left := (i – 1) mod 5; right := i;
    *[      Pfil(left)?in(); -> Pfil(left)?out();
            []
            Pfil(right)?in(); -> Pfil(right)?out();
    ]

ROOM::[ v : integer; v : = 4;
            *[
                        v > 0; (i:0..4) Pfil(i)? ticket() ->
                                                v := v - 1
             []
                        (i:0..4) Pfil(i)?out() ->
                                    v: = v + 1
            ]
]
```

Израчунавање интеграла

Написати програм на језику CSP који израчунава интеграл функције на интервалу XMIN, XMAX у N корака користећи “Торбу Послова“ (Bag of Tasks).

Израчунавање интеграла

```
[Node(p : 1..n) :: NODE || Bag :: BAG]
NODE::[ left, right, data : double;
    *[Bag!getTask(); Bag?getData (left, right) ->
            CALCULATE;
            Bag!putResult(data);
    ]
]
BAG::    [ Xmin, Xmax, dx, x, F : double; F :=0;
    N, i : integer; i := 0;
    INIT;
    *[      i < N, x < Xmax, (j:1..n)Node(j)?getTask() ->[
                        Node(j)!getData (x, x+dx); x := x + dx]
            []
            i < N; (j:1..n)Node(j)?putResult(data) -> [
                        F := F + data; i := i + 1]
    ]
    STOP;
]
```

Питања?

Захарије Радивојевић
Електротехнички Факултет
Универзитет у Београду
zaki@etf.rs