Laborator 6

CPN Tools

- Editor si simulator pentru retele Petri Colorate

- Foloseste un limbaj functional pentru expresiile de arce: **CPN ML**, extensie a limbajului functional **Standard ML**
  - Executia unui program functional are drept scop evaluarea de expresii
  - Nu exista instructiuni care sa modifice zone de momorie sau instructiuni de control!
  - Functiile scrise intr-un limbaj functional sunt asemanatoare functiilor matematice!
  - Nu exista instructiuni de asignare! O variabila este asemanatoare unei constante dintr-un limbaj imperativ de programare (reprezinta un nume pentru pentru o valoare)
  - Declaratii de variabile in Standard ML:
    `val id = valoare;` (id nu mai poate avea alta valoare)
  - Specfice CPN ML sunt declaratii de forma
    `var id: Tip;`
    dar id poate fi folosit doar in expresiile de pe arcele/functiile garda din retea

  - Fiecare expresie trebuie sa aiba un tip (tipul nu trebuie precizat explicit): la compilare se va face o inferenta a tipului expresiei (type inference)
  - Functiile pot fi polimorfice
  - CPN ML extinde Standard ML cu tipuri multiset si operatii specifice

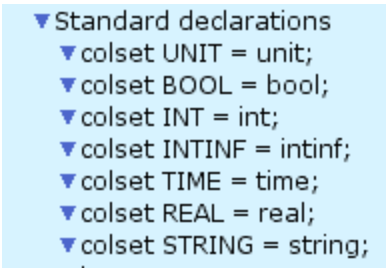
Tipuri in CPN Tools (selectie):

I Tipuri simple (nestructurate)

Tipuri de baza din Standard ML:

- int (atentie, valorile negative sunt ~1, ~2….)
- string (a se vedea documentatia pentru operatii specfice)
- bool (true, false), operatori booleeni: **orelse, andalso, not, <>, =**
- **unit : valoarea (),** nu are operatori specfici, poate fi folosit in multiseturi
- real

CPN ML defineste alias-uri pentru tipurile de baza (nu lucreaza direct cu acestea! Nu se poate defini var x: int):



colset este cuvant rezervat, folosit pentru definirea de noi tipuri
Un alias al unui tip existent: `colset TipNou = TipExistent;`

Tipul enumerare:
`colset Tip = id1|id2….|idn;`
unde id1,...idn sunt identificatori; Valorile acestui tip sunt id1,....idn.

Tipul index:
`colset Tip = index id with n1..n2;`
id-identificator, n1,n2 numere intregi pozitive, n1 <= n2
Valori: id(n1), id(n1+1).....id(n2)

`colset Proc = index P with 1..3;` (elementele tipului P(1), P(2), P(3))

Pornind de la tipurile de baza, folosind o sintaxa specifica, se pot defini si alte tipuri simple (submultimi ale tipurilor de la care se porneste). A se consulta documentatia CPN Tools!

II Tipuri structurate de date
Tipurile produs, record, lista

Tipul produs:
`colset Tip = product` Tip_1`*`Tip_2`*.....*`Tip_n`;`
Unde $Tip_1, Tip_2, ... Tip_n$ sunt tipuri predefinite
Valoare din acest tip: n-uplu (v1,v2,....vn), cu vi valori din Tip_i
Operator specific: #i (v1,v2,....vn) => vi

Pentru orice tip de date cu mai putin de 100 de elemente (cu numele Tip), exista functii specifice:
`Tip.all()` => mutliset cu toate elementele din tip, fiecare element apare cu coeficientul 1 in multiset
`Tip.ran()` => un element random din multiset

Functii in CPN Tools:

- Repezinta valori de ordin intai (pot fi folosite la fel ca orice alta valoare, aceste valori au un tip special -tipul functional)
- Functiile pot fi polimorfice

Sintaxa:

```
fun Id pattern = expresie;
```

pattern: o expresie mai speciala, poate fi:

- o variabila
- un tuplu care contine variabile, constante, _, sau alte pattern-uri

Intr-un pattern aceeasi variabila nu poate sa apara de mai multe ori.

Evaluarea unei functii: variabilele din pattern se potrivesc/“leaga” de parametrii de apel/evaluare si apoi se evalueaza expr

Exemple:

```
fun sum (x,y) = x + y;
```

Daca se evalueaza in Standard ML aceasta expresie =>
val sum = fn : int * int -> int
(sum este o valoare avand tipul functional fn cu domeniul int * int si codomeniul int)
(x,y) este un pattern = tuplu de variabile, functiile au de fapt un singur parametru!

Evaluarea unei functii cu parametri specfici: sum(2,3) (2,3) pattern matching cu (x,y), x se “leaga” (bind) de 2 si y de 3; apoi se evalueaza 2+3

Se poate defini si:

```
fun sum (x,5) = x+5;
```

(daca se evalueaza sum cu sum(2,6) se va primi o eroare!)

Se poate defini si:

```
fun  sum (x:int,y:int):int = x + y;
```

(se precizeaza explicit un tip pentru elementele din tuplu/pattern, de obicei pentru a impiedica polimorfismul sau in unele cazuri in care dorim ca x sa fie dintr-un anumit tip structurat; `x:int` este o expresie cu adnotare de tip)

Polimorfism:

```
fun mk_ms (i,x) = i`x;
```

(` este apostroful din multiset! Atentie la simbolul folosit, a nu se confunda cu ‘)
Aceasta definitie evaluata in Standard ML produce:

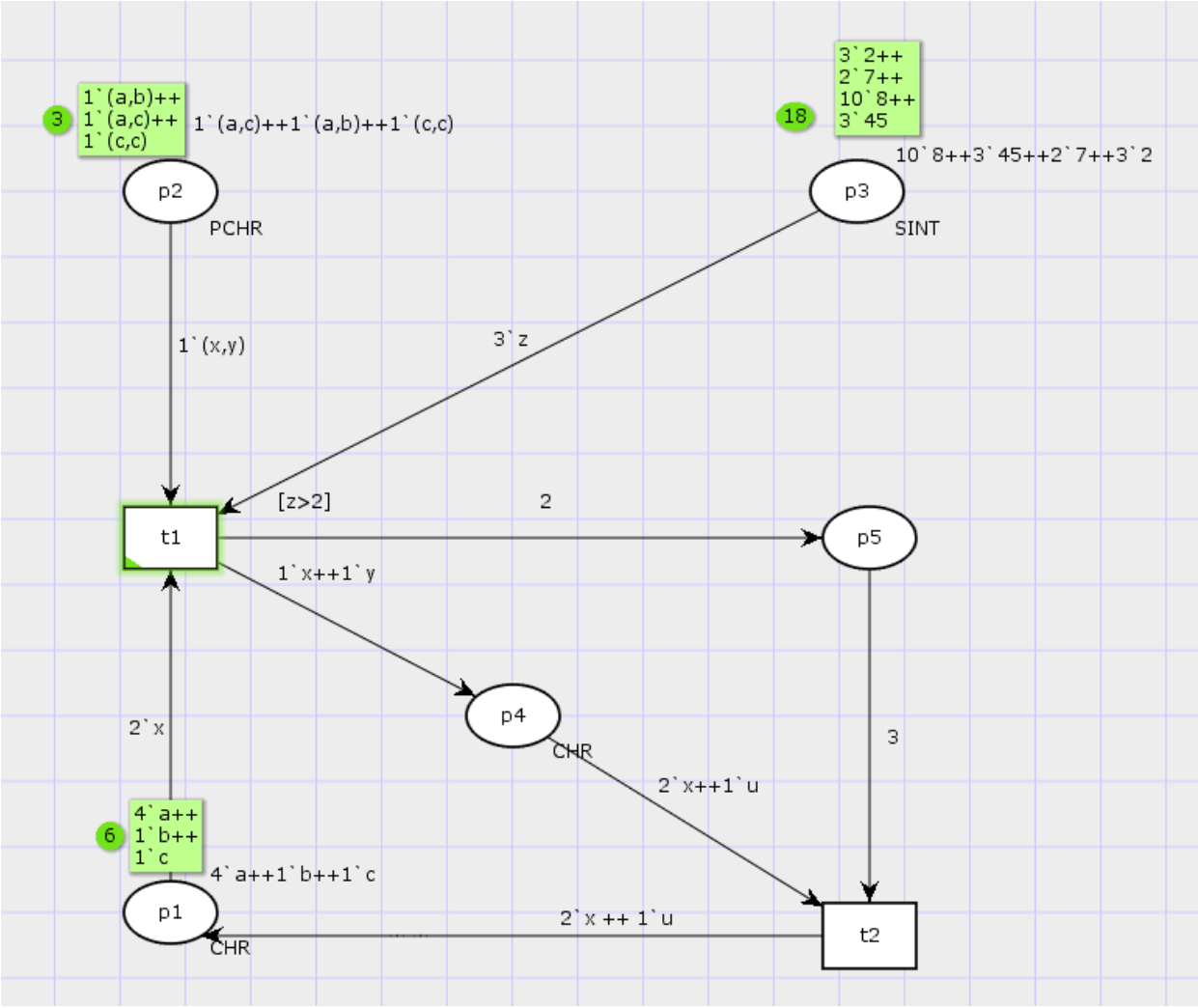
val mk_ms = fn: int * ‘a -> ’a ms

(‘a reprezinta orice tip, ‘a ms este multiset peste tipul ‘a)

mk_ms(2,4) se va evalua la 2`4
mk_ms(2,”zzz”) se va evalua la 2`”zzz”

```
fun sum5(_,x) = x + 5;
```
(_ reprezinta orice)
val sum5: ‘a * int -> int

```
fun identity(_,x) = x;
```
val identity: ‘a * ‘b -> ‘b (‘a este un tip oarecare, ‘b este un tip oarecare, cele doua nu sunt distincte in mod obligatoriu)



completati declaritiile
editati si simulati reteaua

Ex2:
Problema cu procese si resurse din cursul 5 (slide 27)

Se va folosi un tip index pentru tipul proceselor/resurselor

```
val n=3;
colset Proc = index P with 1..n;
colset Res = index R with 1..2;
```

Atentie la notatia pentru multiset! Folositi sintaxa cu ++ din CPN ML si apostroful corespunzator!

O expresie "if" pe arce:

```
if expr_booleana then expr1 else expr2
```

`expr1` si `expr2` sunt expresii care trebuie sa aiba un tip comun `Tip`
Intreaga expresie are tipul `Tip`.
`else` nu este optional!

```
if x <= 3 andalso y = "ana" then 10 else 20
```

Expresia are tipul int si poate fi folosita in orice alta expresie cu tip int:

```
3 + x + if x <= 3 andalso y = "ana" then 10 else 20 + 5
```

(A se vedea si expresii de tip case in documentatie!)
Pentru a nu repeta expresia "if…" pe 2 arce, folositi o functie:
<u>Varianta 1:</u>

```
fun getRes(x) = if x= P(1) orelse x = P(2) then 1`R(1)++1`R(2)
else 1`R(2);
```

<u>Varianta 2:</u>

```
fun getRes(P(i)) = if i = 1 orelse i = 2 then 1`R(1)++1`R(2)
else 1`R(2);
```

<u>Varianta 3:</u>

```
fun getRes(P(i)) = case i of 1 => 1`R(1)++1`R(2)
                          | 2 => 1`R(1)++1`R(2)
                          | 3 =>  1`R(2);
```

Sau:

```
fun getRes(P(i)) = case i of 3 => 1`R(2)
```

```
  | _ => 1`R(1)++1`R(2);
```