Développement de Taylor des programmes informatiques

Christine Tasson

Laboratoire Preuves Programmes Systèmes - Université Paris Diderot

4 juin 2008

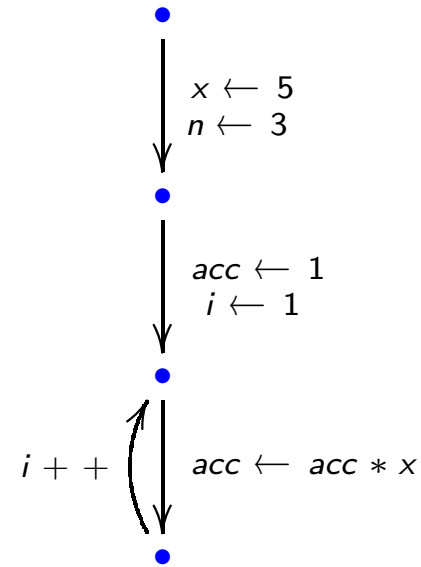
Plan de l'exposé

Les mains dans le cambouis

On considère le programme qui calcule la fonction :

$$(x, n) \mapsto x^n + 1$$

```
public class Puissance{
  public static void main(String[] args){
    int n=args[0];
    int x=args[1];
    println(puissance(n,x)+1);
  }
  public static int puissance(int n, int x){
    int acc=1;
    for (i=1 ; i<=n ; i++){
      acc = acc * x;
    }
    return acc;
  }
}
```



Questions :

- Comment le résultat est-il calculé ?
- Combien de fois utilise-t-on chacun des arguments ?

La tête dans l'espace

Les programmes sont modélisés par des fonctions :

$$[Prog] : A \to B$$

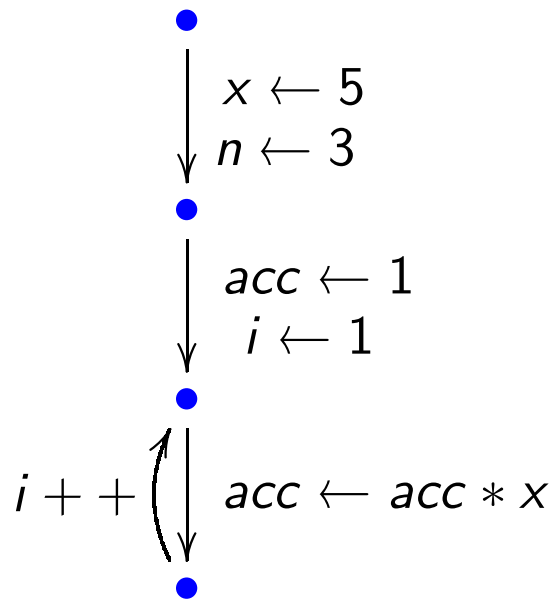
Si le modèle est assez précis, il reflète les propriétés du programme.

Le modèle vectoriel :

- arguments, résultats : espaces vectoriels
- programmes : fonctions linéaires

La linéarité, une analogie

- Une fonction est dite linéaire si son résultat est proportionnel à son argument.
- Un programme est dit linéaire s'il utilise une *seule* fois son argument au cours de son évaluation.



Comment coder un programme qui n'est pas linéaire ?

On le décompose en :

- une partie linéaire $\multimap$
- une partie exponentielle : !

[Puissance] :

$$\begin{array}{ccccc} !X & \times & N & \multimap & R \\ (x & , & n) & \mapsto & x^n \end{array}$$

L'enseignement des mathématiques

Comment faire une approximation linéaire d'un programme ?

En mathématiques, le calcul différentiel nous permet d'approcher certaines fonctions par des fonctions linéaires :

$$f(x) \underset{x \to 0}{\simeq} f(0) + f'(0)\, x$$

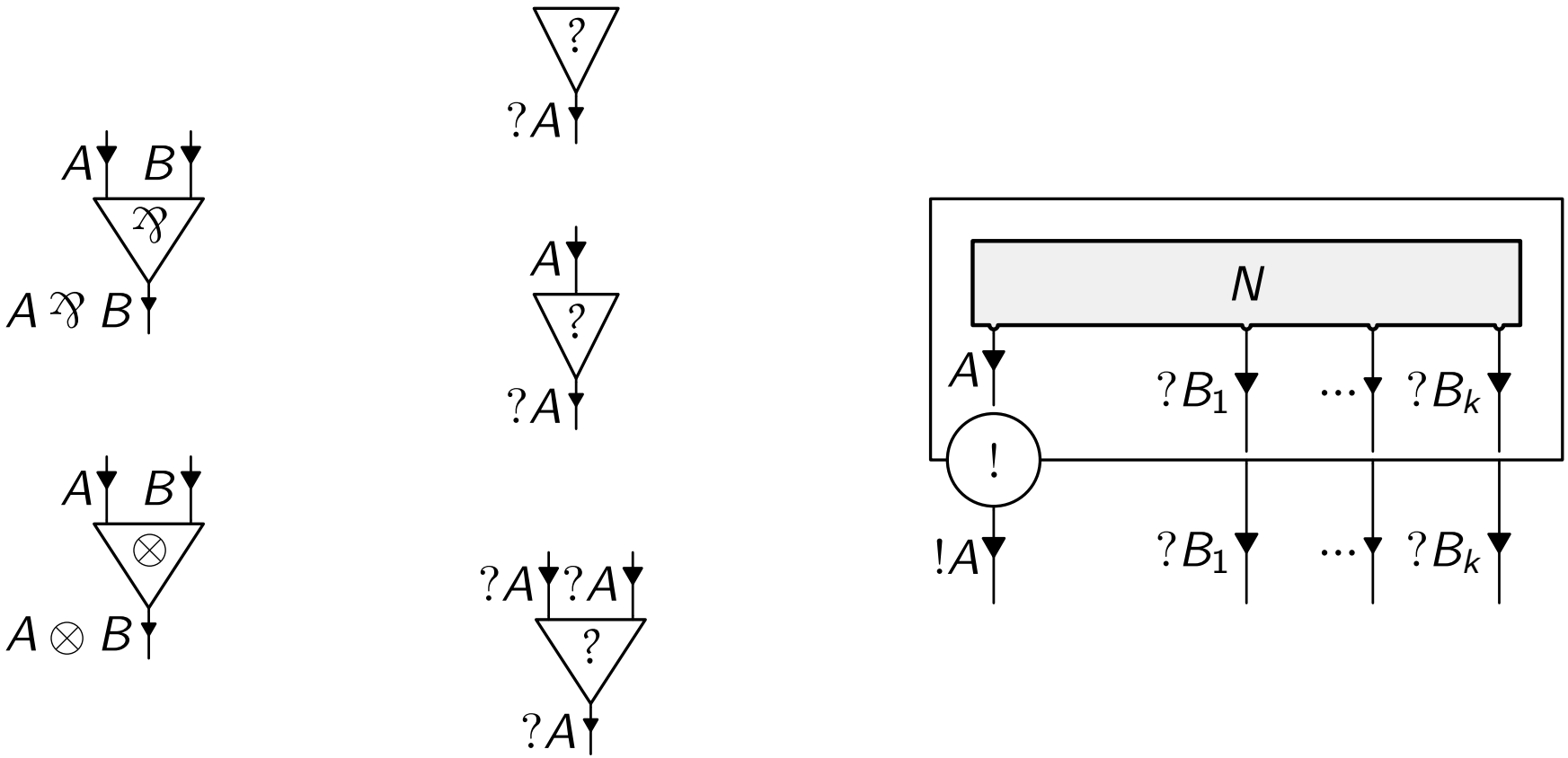
On peut même décomposer toute fonction analytique en une somme de fonctions n-linéaires :

$$f(x) = \sum_n \frac{f^{(n)}(0)}{n!}\, x^n$$

La formule de Taylor se généralise aux programmes.

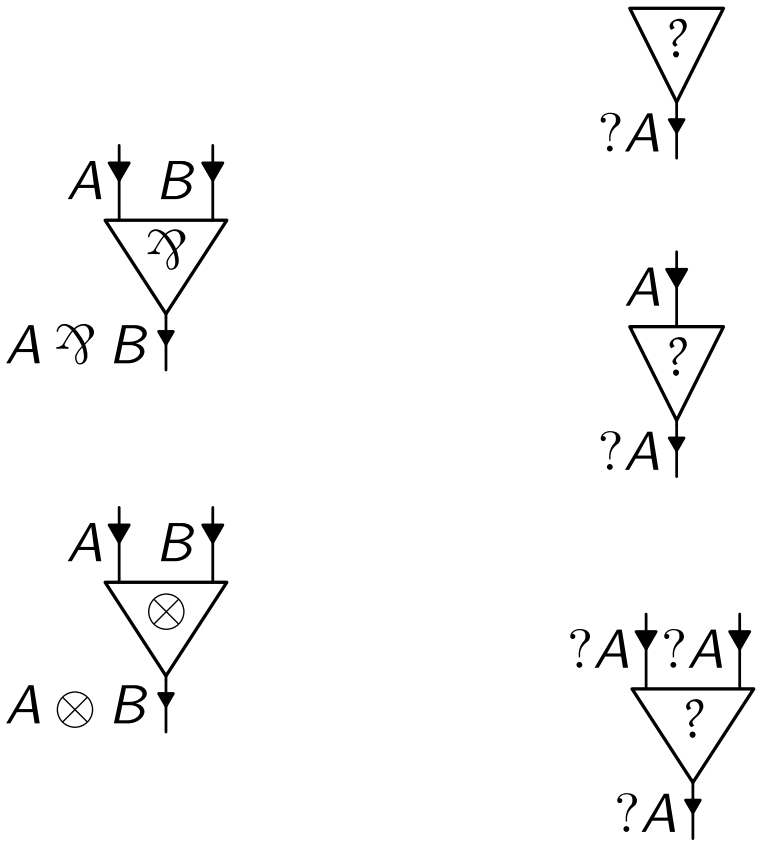
Réseaux de la logique linéaire

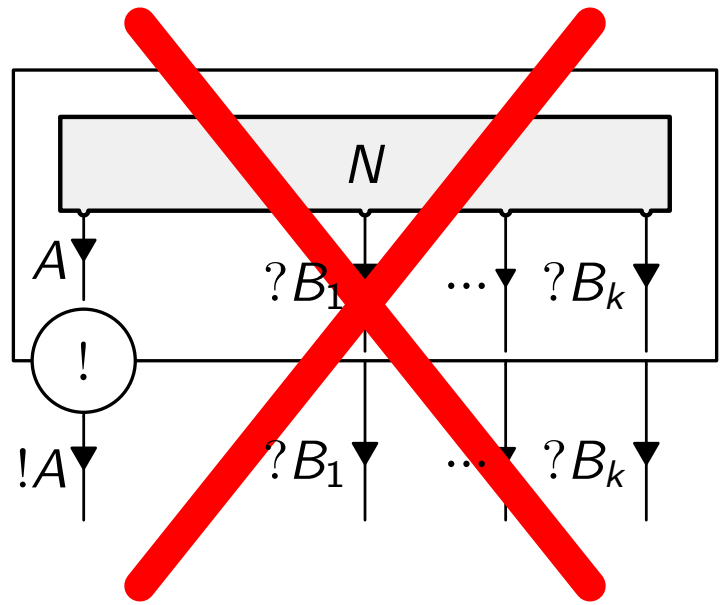
Description du langage de programmation



Réseaux différentiels

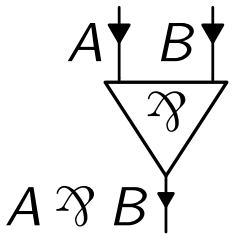
Description du langage linéarisé :

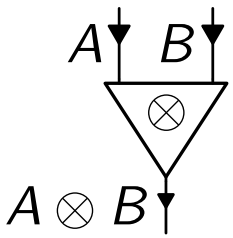


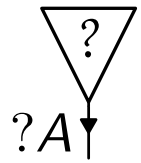


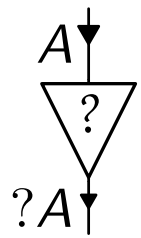
Réseaux différentiels

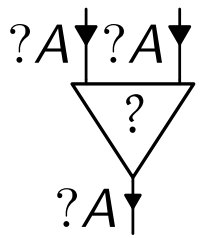
Description du langage linéarisé :

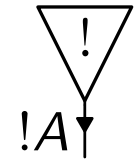


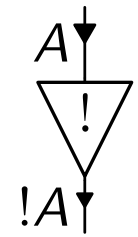


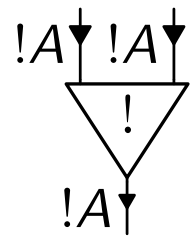






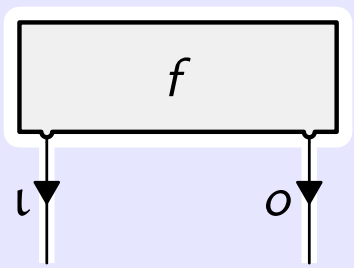




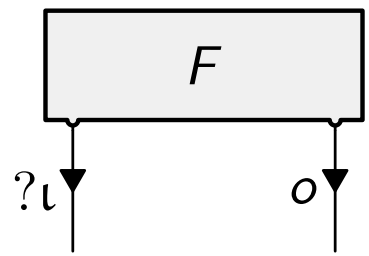


Réseaux et fonctions mathématiques

Fonction linéaire :

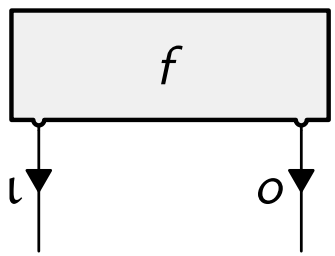


Fonction analytique :

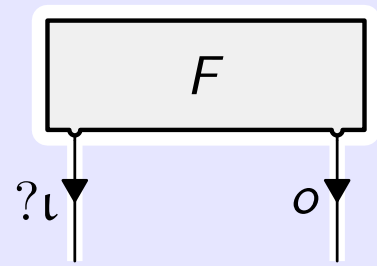


Réseaux et fonctions mathématiques

Fonction linéaire :

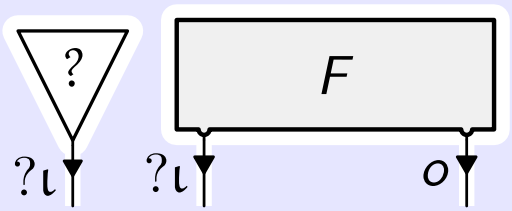


Fonction analytique :



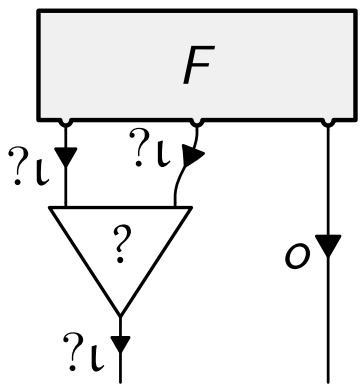
Réseaux et fonctions mathématiques (1/3)

$$G : x, y \mapsto F(x)$$



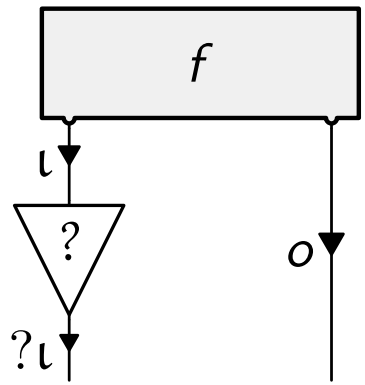
Affaiblissement

$$G : x \mapsto F(x, x)$$



Contraction

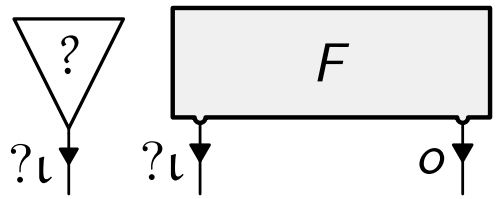
$$G : x \mapsto f(x) + \sum 0 \times \frac{x^n}{n!}$$



Déréliction

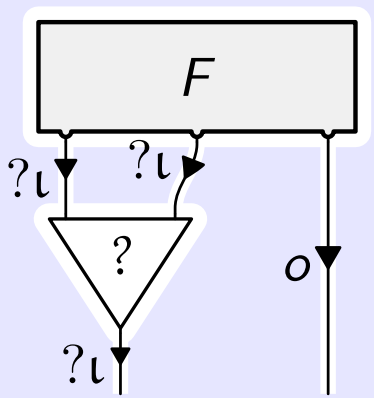
Réseaux et fonctions mathématiques (1/3)

$G : x, y \mapsto F(x)$



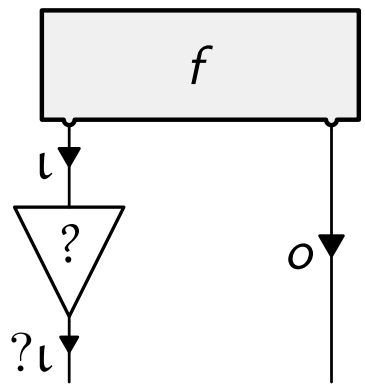
Affaiblissement

$G : x \mapsto F(x, x)$



Contraction

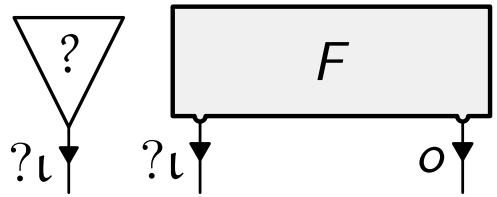
$G : x \mapsto f(x) + \sum 0 \times \frac{x^n}{n!}$



Déréliction

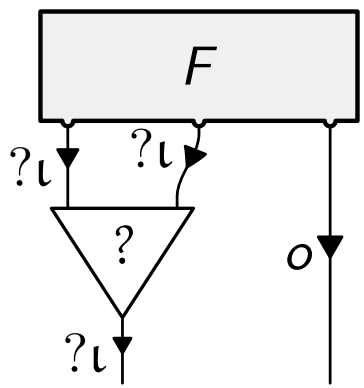
Réseaux et fonctions mathématiques (1/3)

$G : x, y \mapsto F(x)$



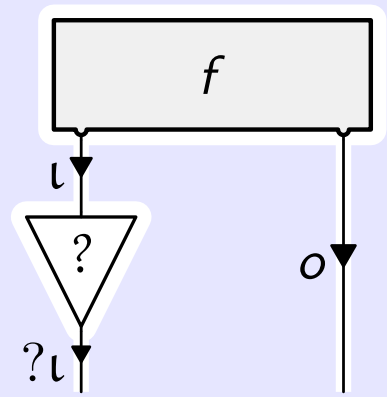
Affaiblissement

$G : x \mapsto F(x, x)$



Contraction

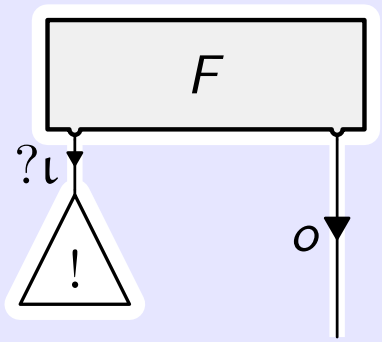
$$G : x \mapsto f(x) + \sum 0 \times \frac{x^n}{n!}$$



Déréliction

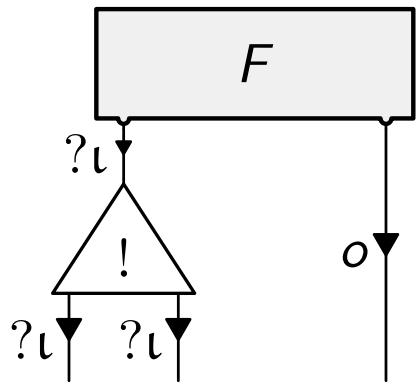
Réseaux et fonctions mathématiques (2/3)

$G : x \mapsto F(0)$



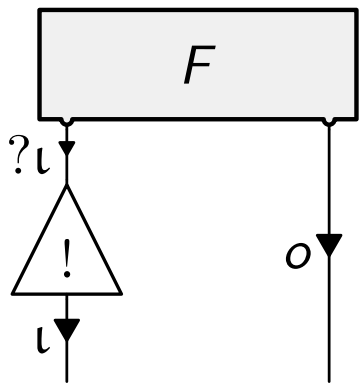
Co-affaiblissement

$G : x, y \mapsto F(x + y)$



Co-contraction

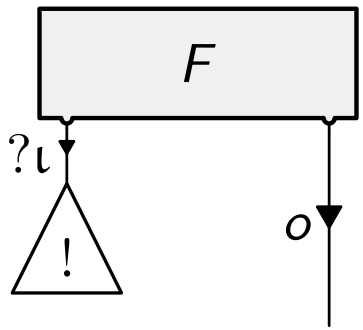
$g : x \mapsto F'(0) * x$



Co-déréliction

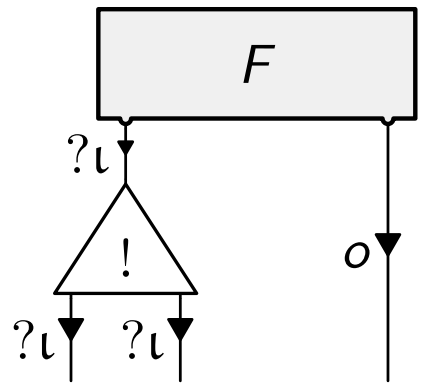
Réseaux et fonctions mathématiques (2/3)

$G : x \mapsto F(0)$



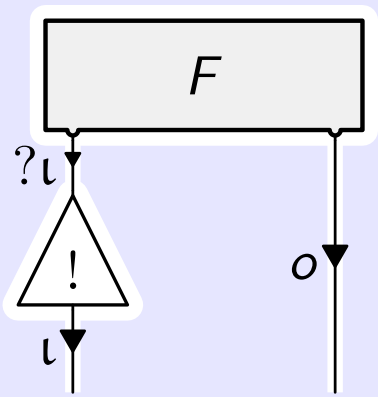
Co-affaiblissement

$G : x, y \mapsto F(x + y)$



Co-contraction

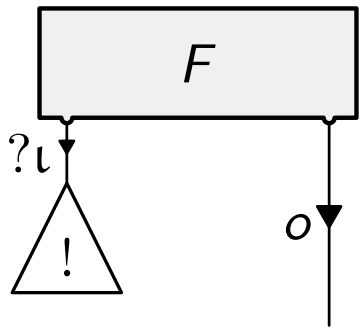
$g : x \mapsto F'(0) * x$



Co-déréliction

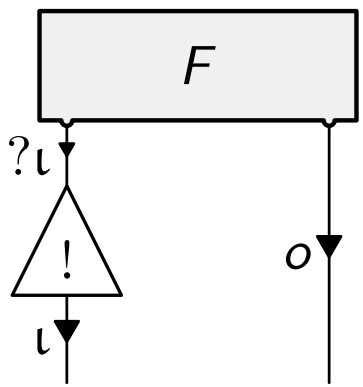
Réseaux et fonctions mathématiques (2/3)

$G : x \mapsto F(0)$



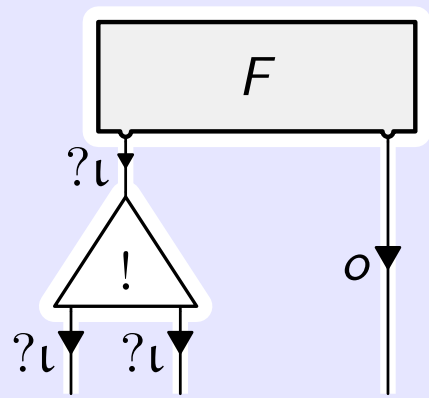
Co-affaiblissement

$g : x \mapsto F'(0) * x$



Co-déréliction

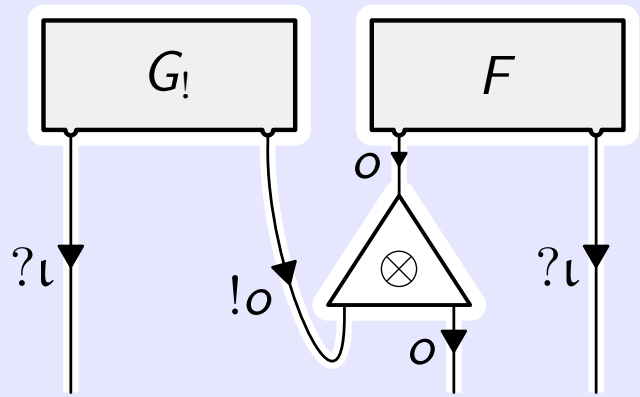
$G : x, y \mapsto F(x + y)$



Co-contraction

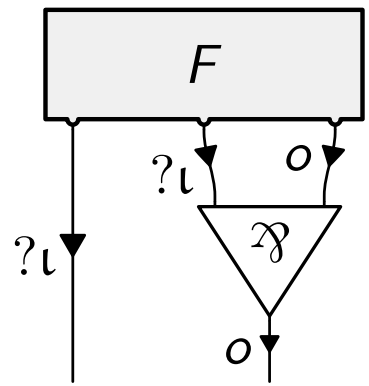
Réseaux et fonctions mathématiques (3/3)

$x, y \mapsto F(G(x), y)$



Tenseur

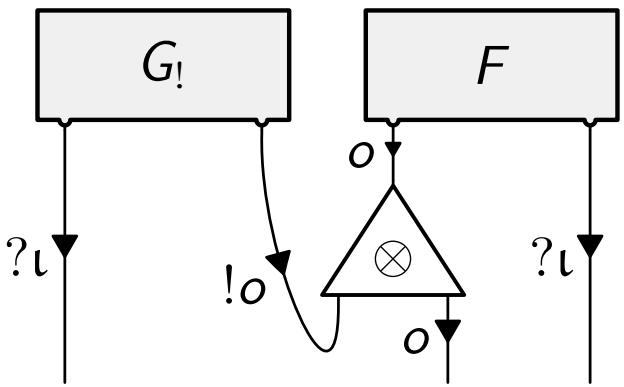
$x \mapsto [y \mapsto F(x, y)]$



Par

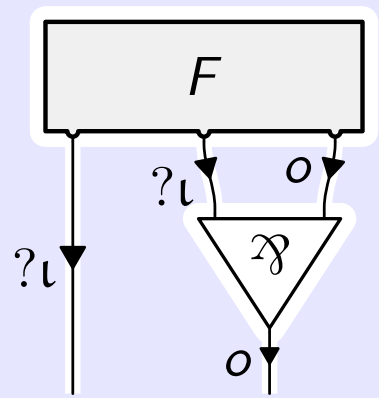
Réseaux et fonctions mathématiques (3/3)

$x, y \mapsto F(G(x), y)$



Tenseur

$x \mapsto [y \mapsto F(x, y)]$

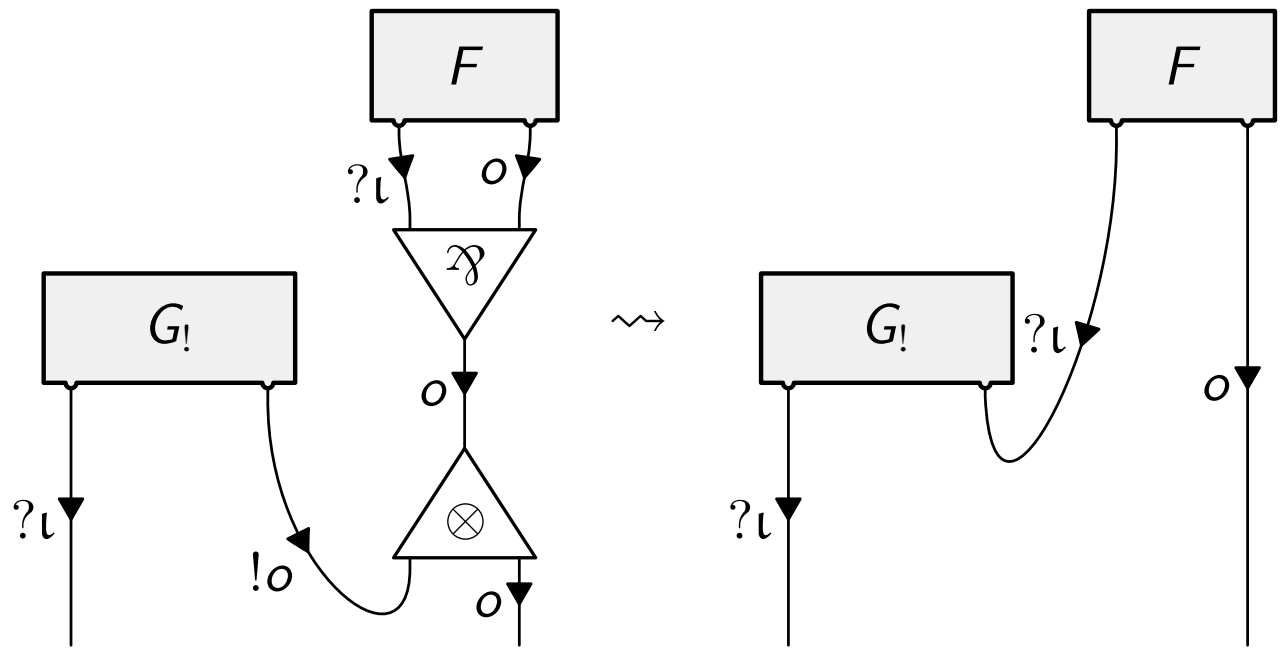


Par

Règles de calcul (1/3)

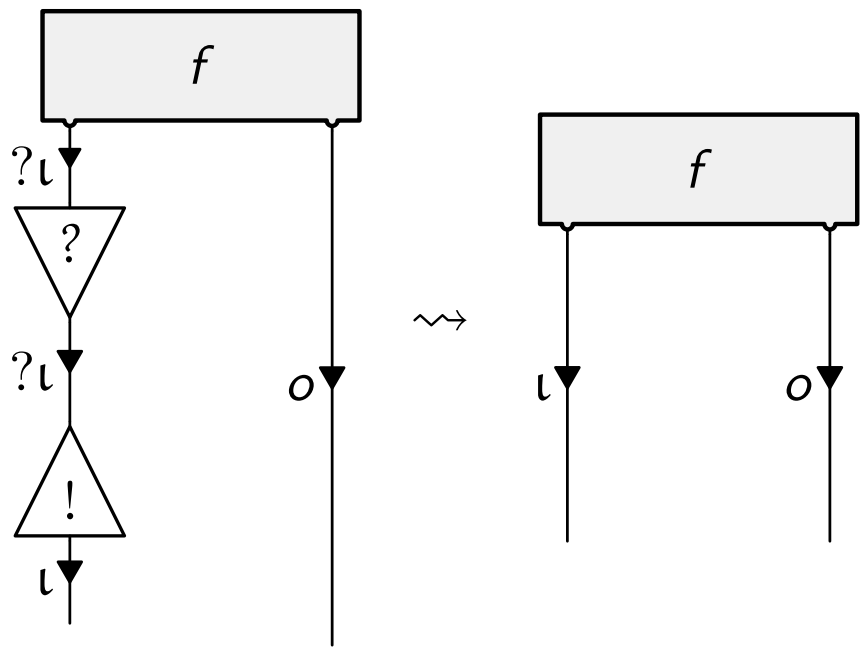
Tenseur/Par :

$$F(G(X)) = F \circ G(X)$$



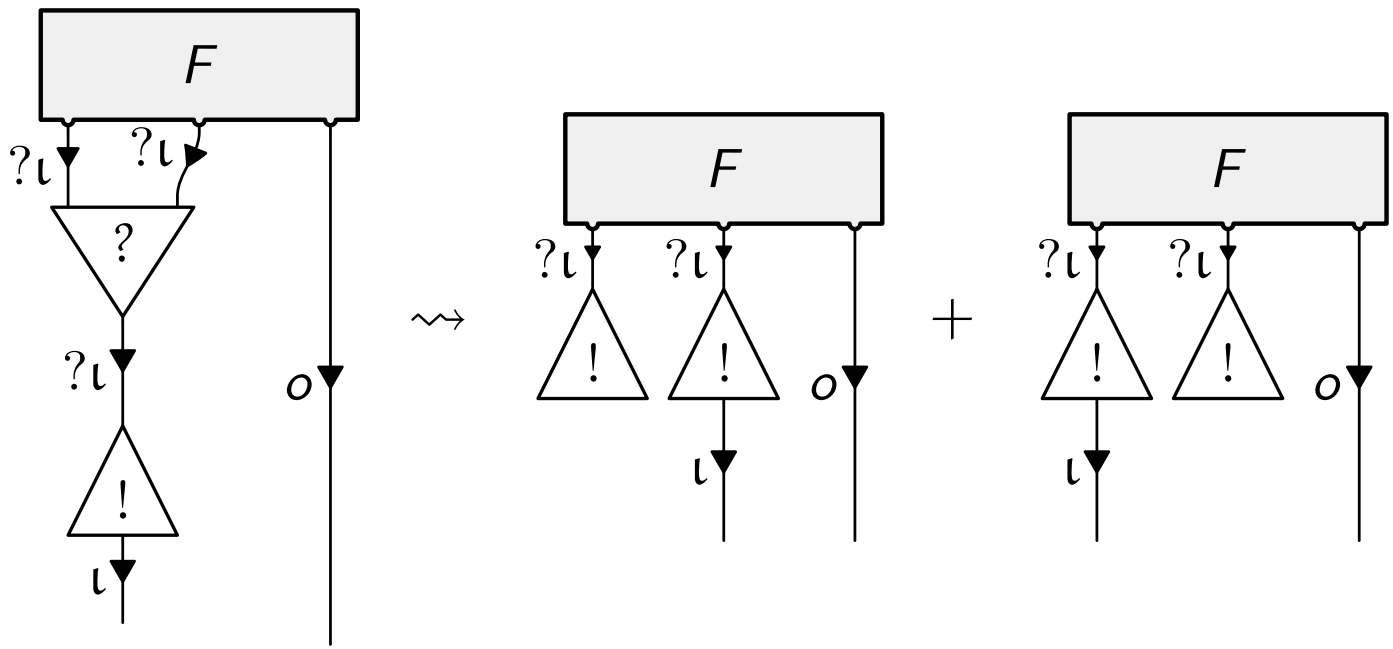
Règles de calcul (2/3)

Linéarité :
Si f est linéaire, alors $Df = f$.



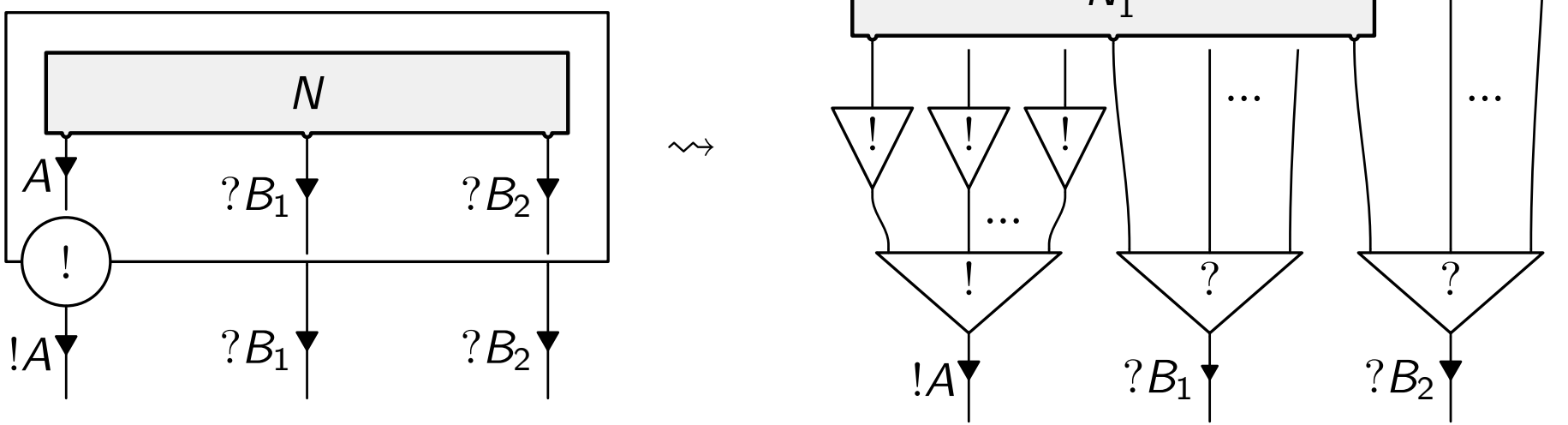
Règles de calcul (3/3)

Contraction/Codéréliction :
$D_0(x \mapsto F(x,x)) = \partial_1 F(0,0) + \partial_2 F(0,0)$



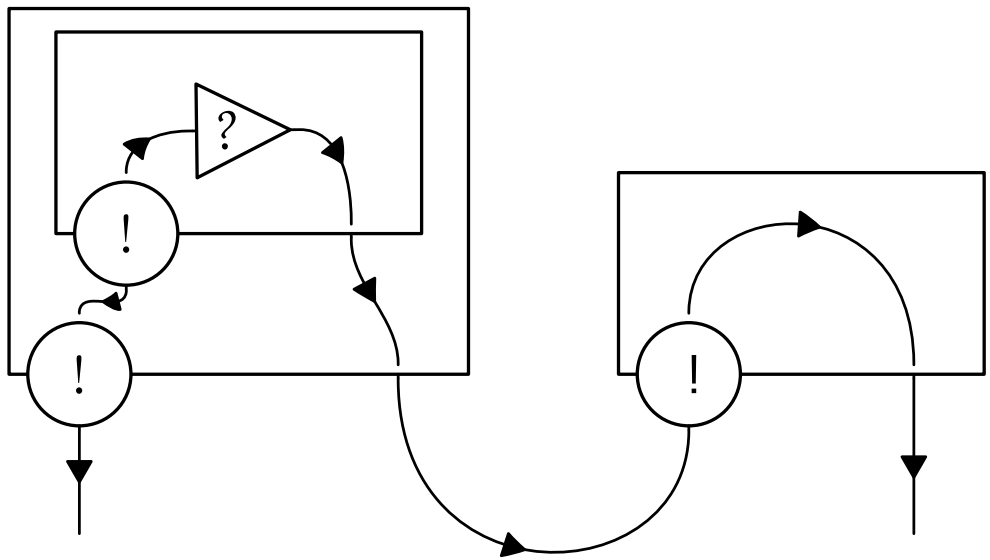
Taylor s'invite dans l'informatique

Le principe de base : à chaque réseau de la logique linéaire et pour tout n, on associe un réseau différentiel.



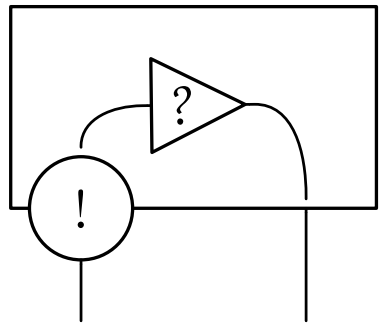
où N_k^* apparaît dans le développement de Taylor de N.

Taylor sur un exemple

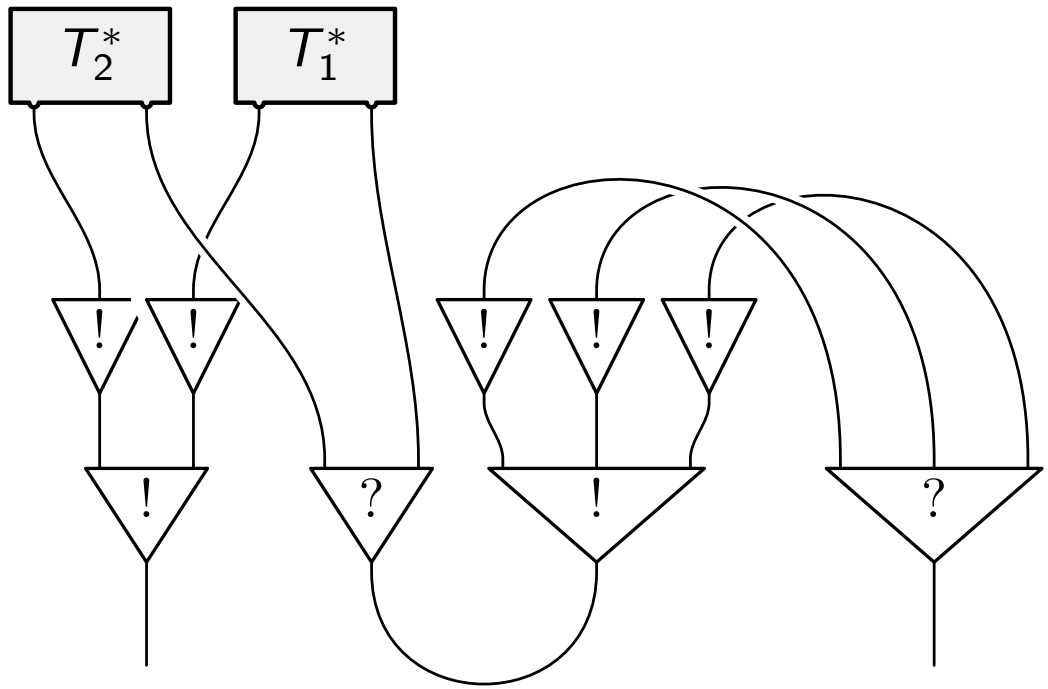


Taylor sur un exemple

T :

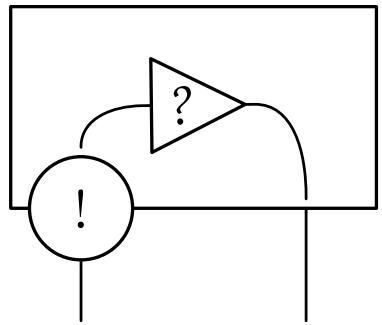


où T_1^* et T_2^* apparaissent dans le développement de Taylor de T.

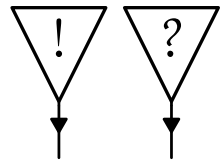


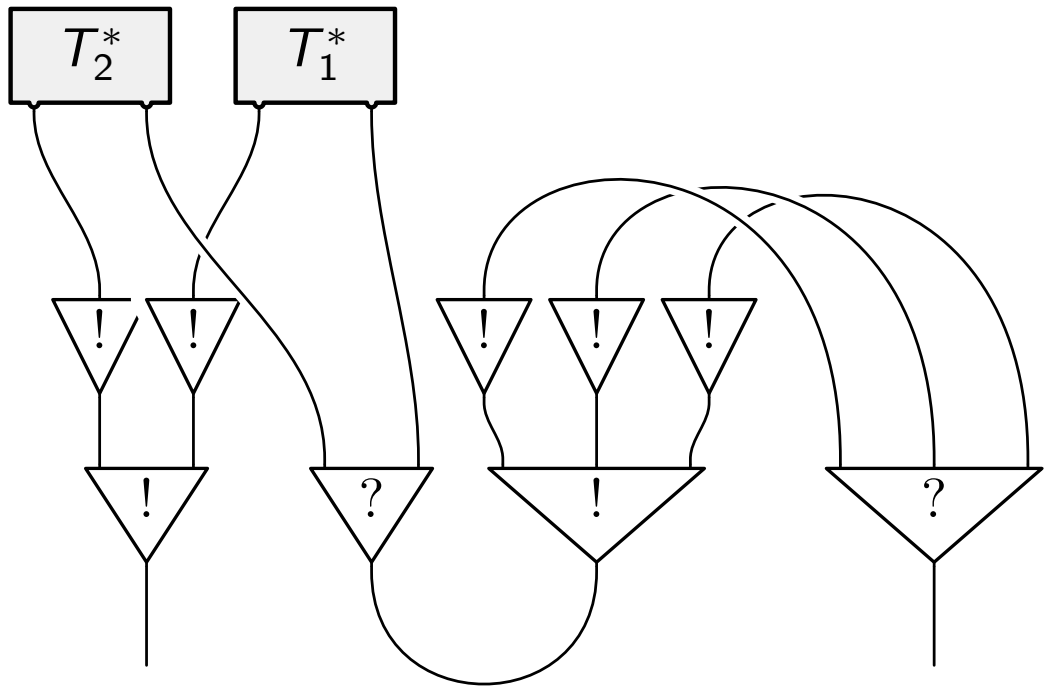
Taylor sur un exemple

T :



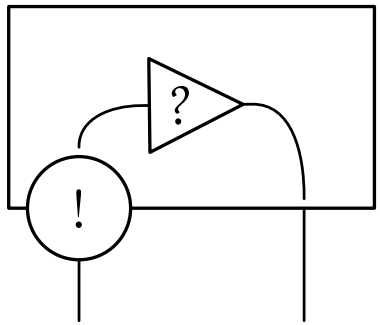
$\rightsquigarrow T_1^*$:



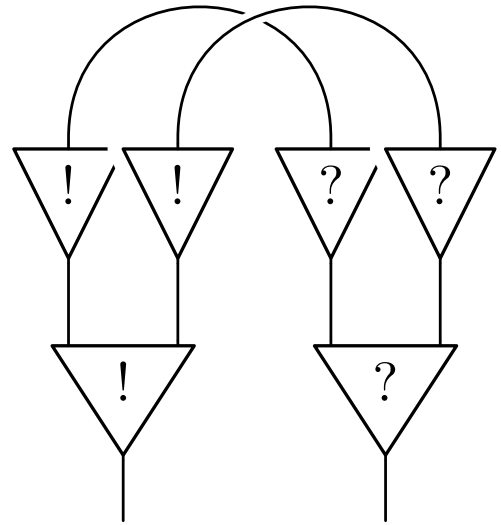


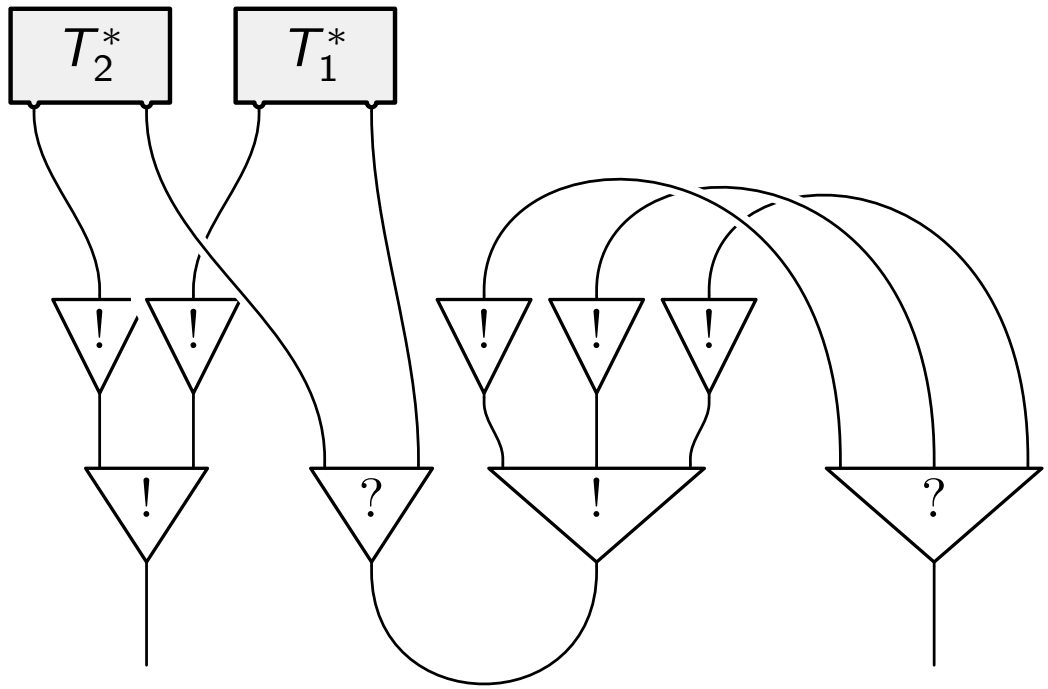
Taylor sur un exemple

T :

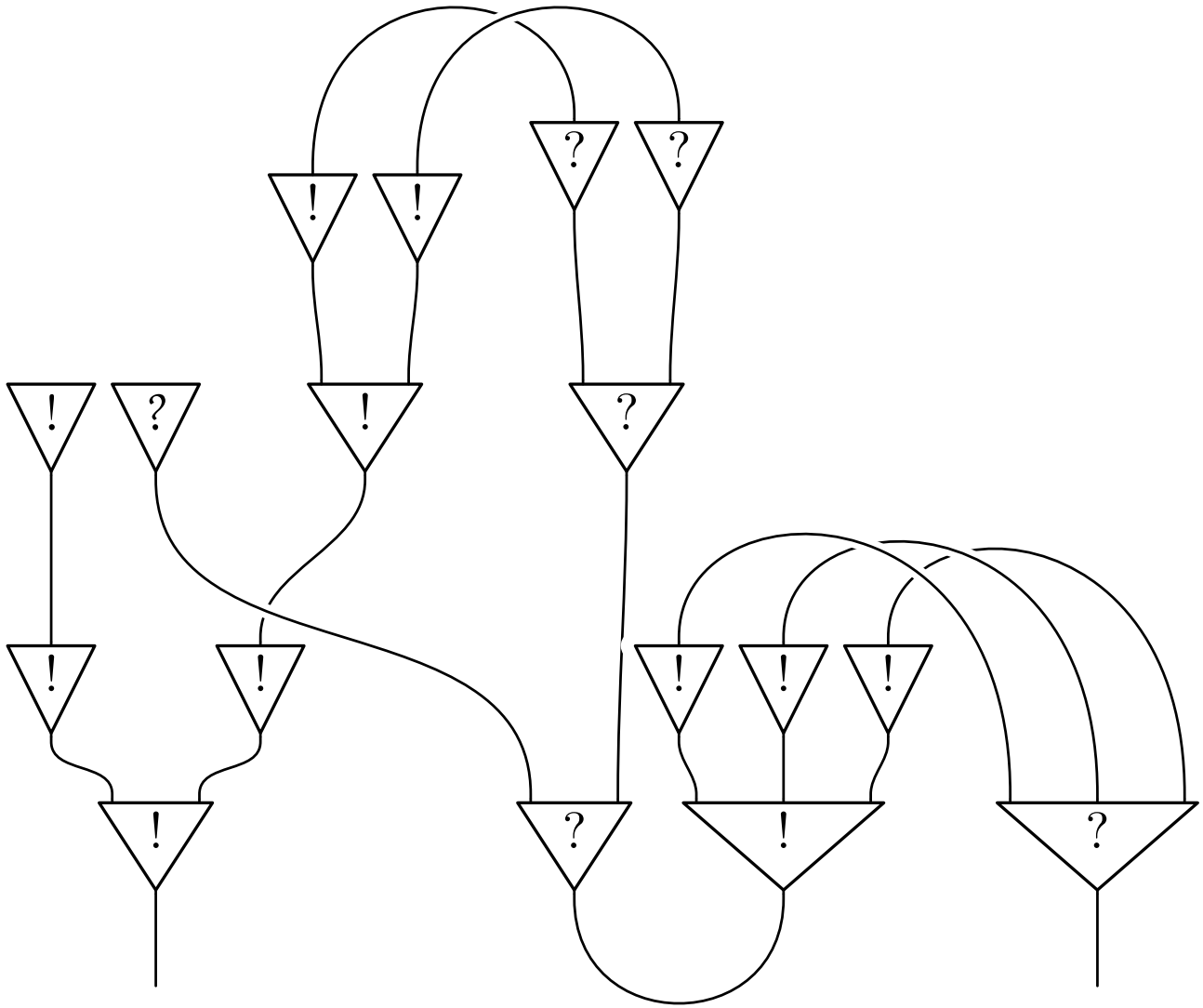


$\rightsquigarrow T_2^*$:

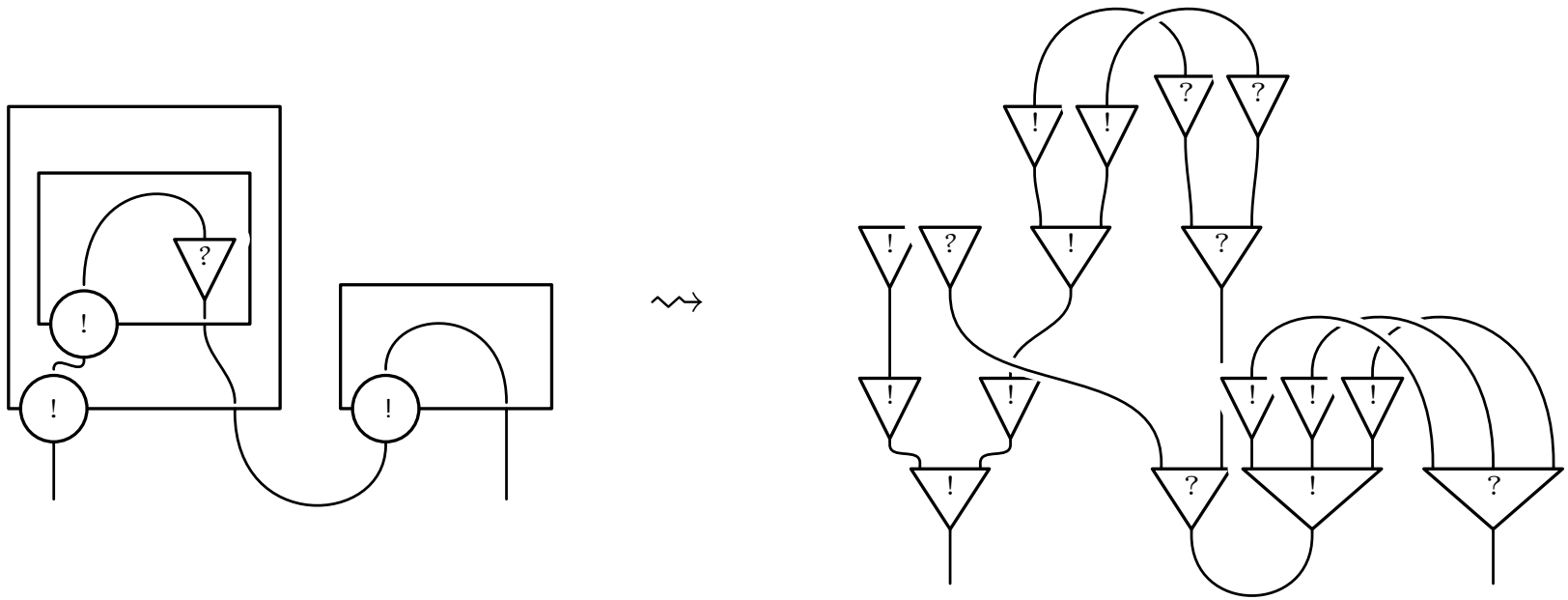




Taylor sur un exemple



Taylor sur un exemple



Taylor s'invite dans l'informatique

Rappel : la formule de Taylor

$$f(x) = \sum_n \frac{f^{(n)}(0)}{n!} x^n$$

Théorème

Tout programme P peut s'écrire sous la forme suivante :

$$P = \sum_{p \in \mathcal{T}(P)} c(P, p) \cdot p$$

Interprétation :

- p est une version de P qui utilise exactement n fois son argument.
- $c(P, p)$ nombre de façon d'obtenir le terme p.
- $\mathcal{T}(P)$ ensemble des approximations n-linéaires de P.

Taylor s'invite dans l'informatique

Rappel : la formule de Taylor

$$f(x) = \sum_n \frac{f^{(n)}(0)}{n!} x^n$$

Théorème

Tout programme P peut s'écrire sous la forme suivante :

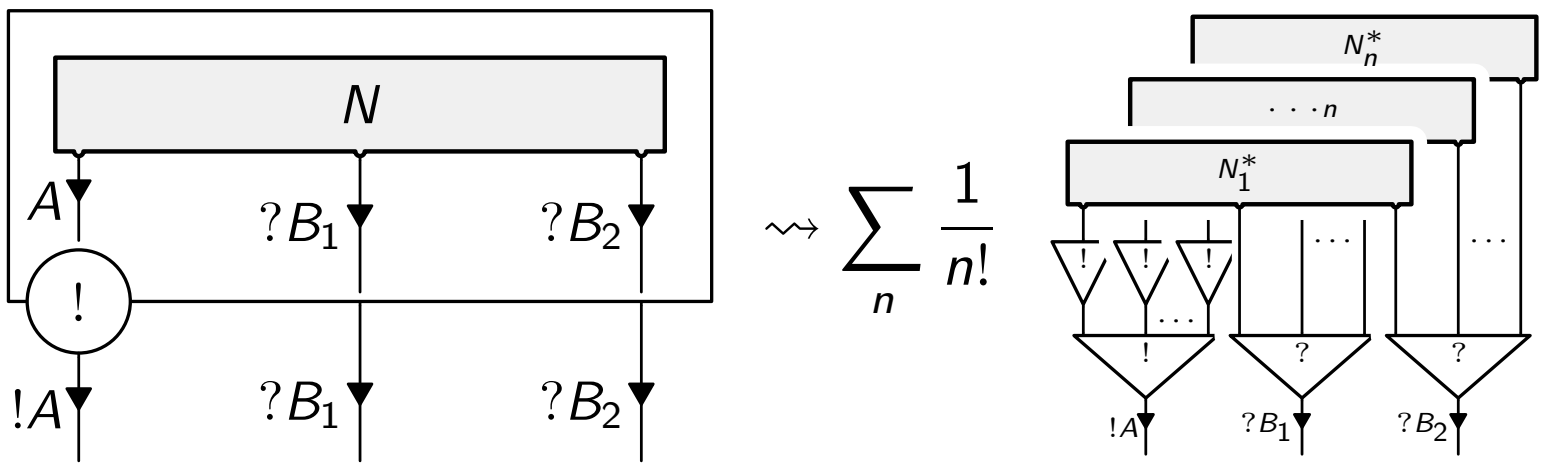
$$P = \sum_{p \in \mathcal{T}(P)} c(P, p) \cdot p$$

Questions :

- Comment calculer les coefficients ?
- Montrer que $c(P, p)$ ne dépend que de p.

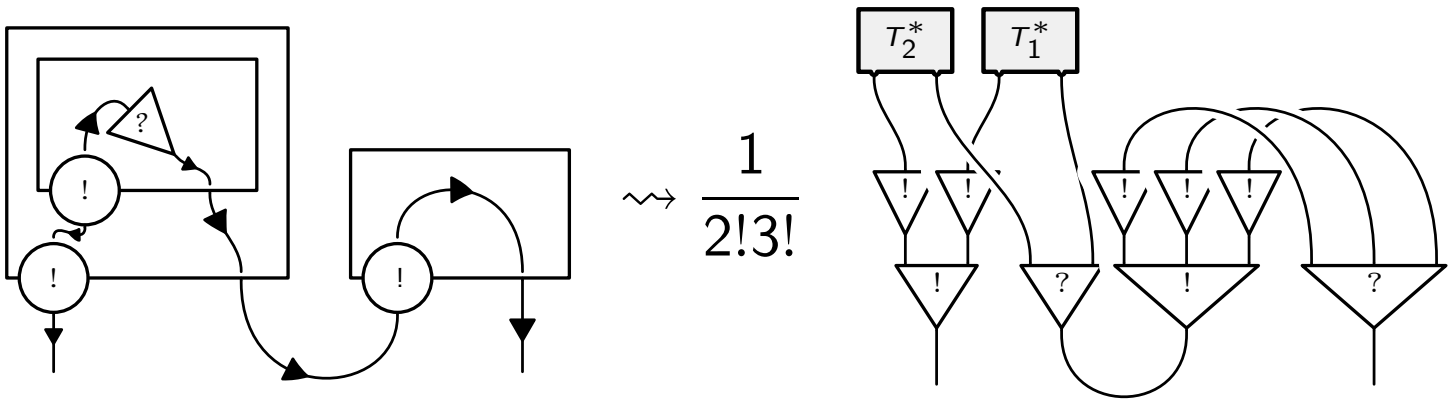
Coefficients, le principe

A chaque boîte du réseau et à chaque n on associe un réseau différentiel et un coefficient qui correspond au nombre de réseaux équivalents modulo commutativité des contractions et co-contractions.

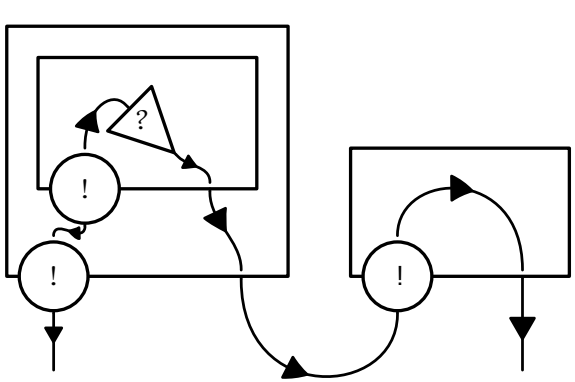


où N_k^* apparaît dans le développement de Taylor de N.

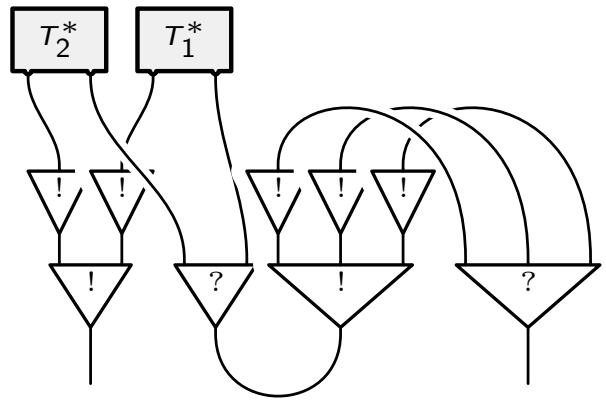
Coefficients, l'exemple



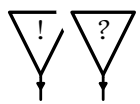
Coefficients, l'exemple



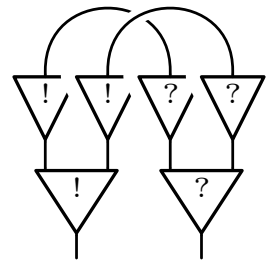
$\rightsquigarrow \frac{1}{2!3!}$



T_1^* : $\frac{1}{0!}$

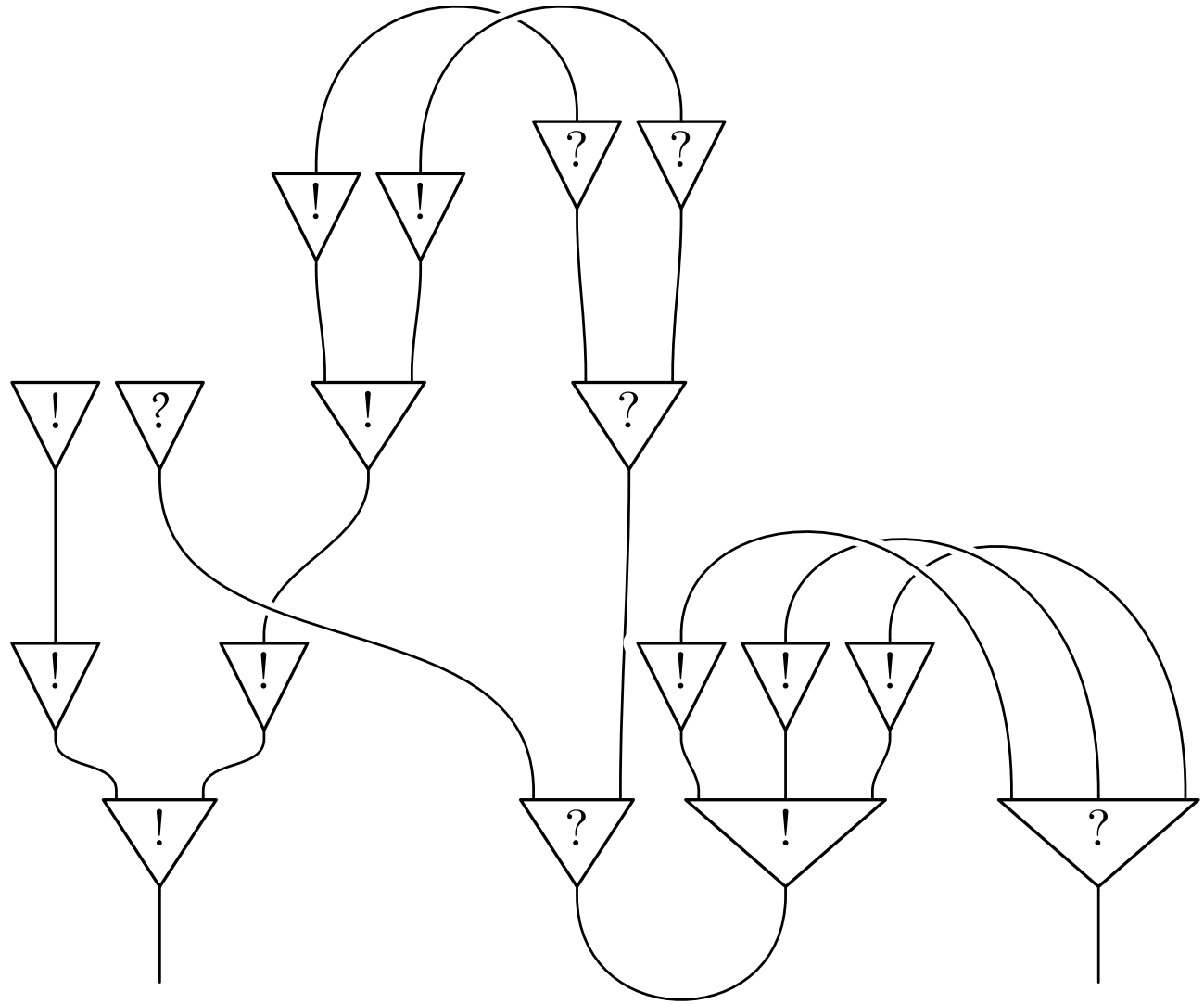


T_2^* : $\frac{1}{2!}$

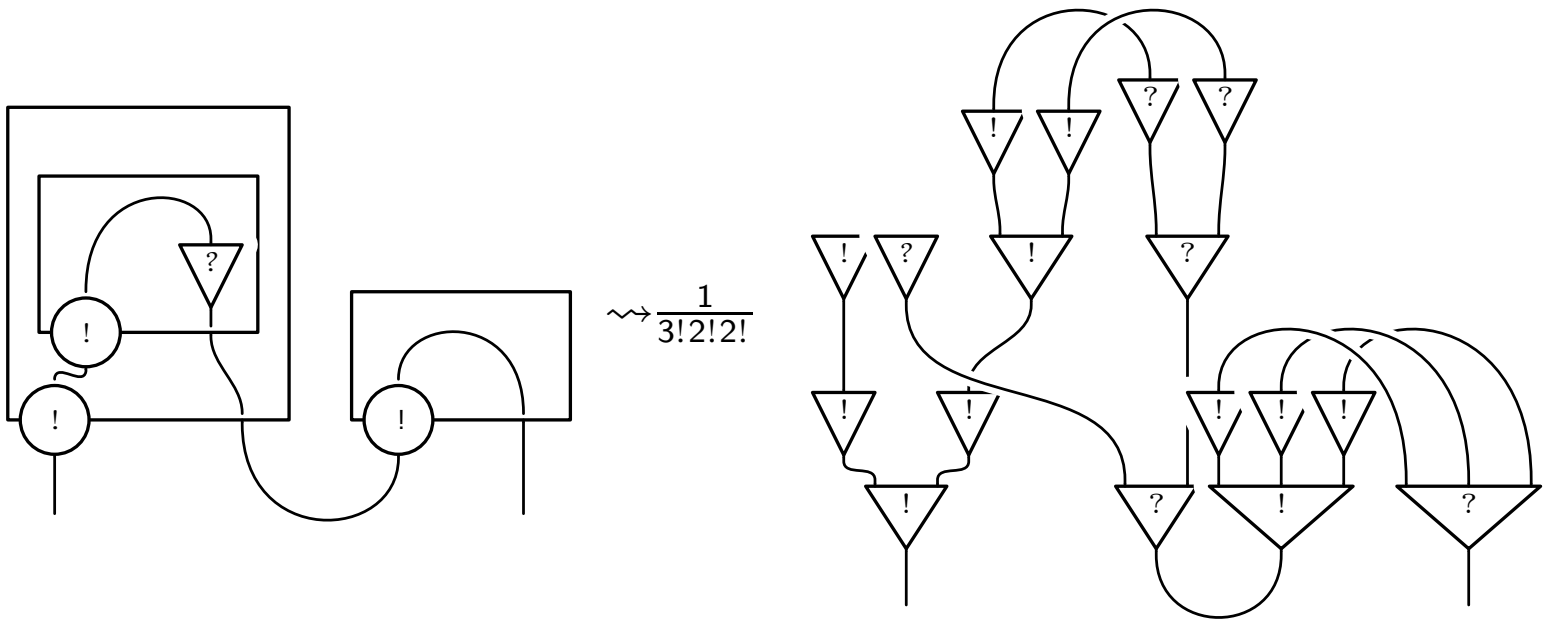


Coefficients, l'exemple

$$\frac{1}{3!2!2!}$$



Coefficients, l'exemple



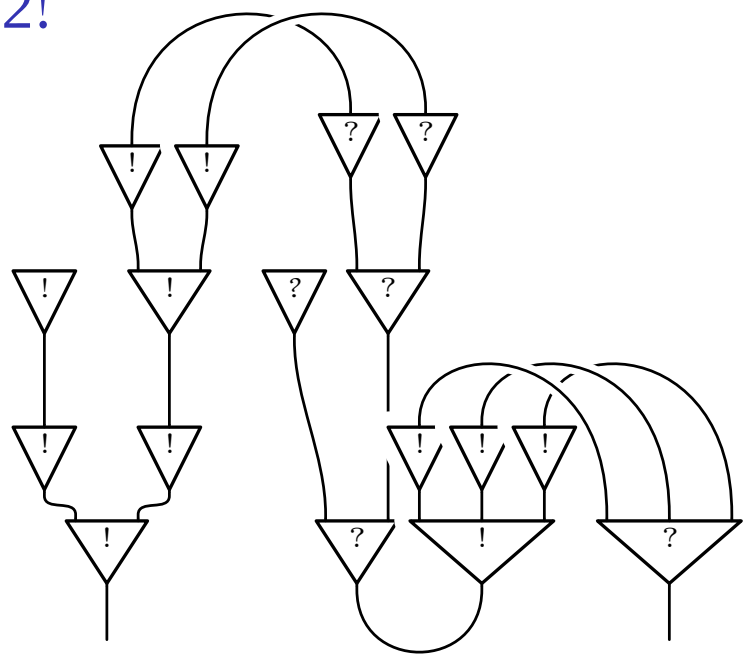
Le coefficient 3!2!2! ressemble fortement au cardinal d'une orbite sous l'action d'un groupe de permutation.
Mais, lequel ?

Le groupe d'isotropie

Calcul des coefficients

Soit p un réseau différentiel apparaissant dans le développement de Taylor d'un réseau de la logique linéaire. Le coefficient $c(p)$ est lié au nombre de réseaux isomorphes à p modulo permutation des arbres de co-contraction et des arbres de contraction.

Un exemple : 3!2!2!

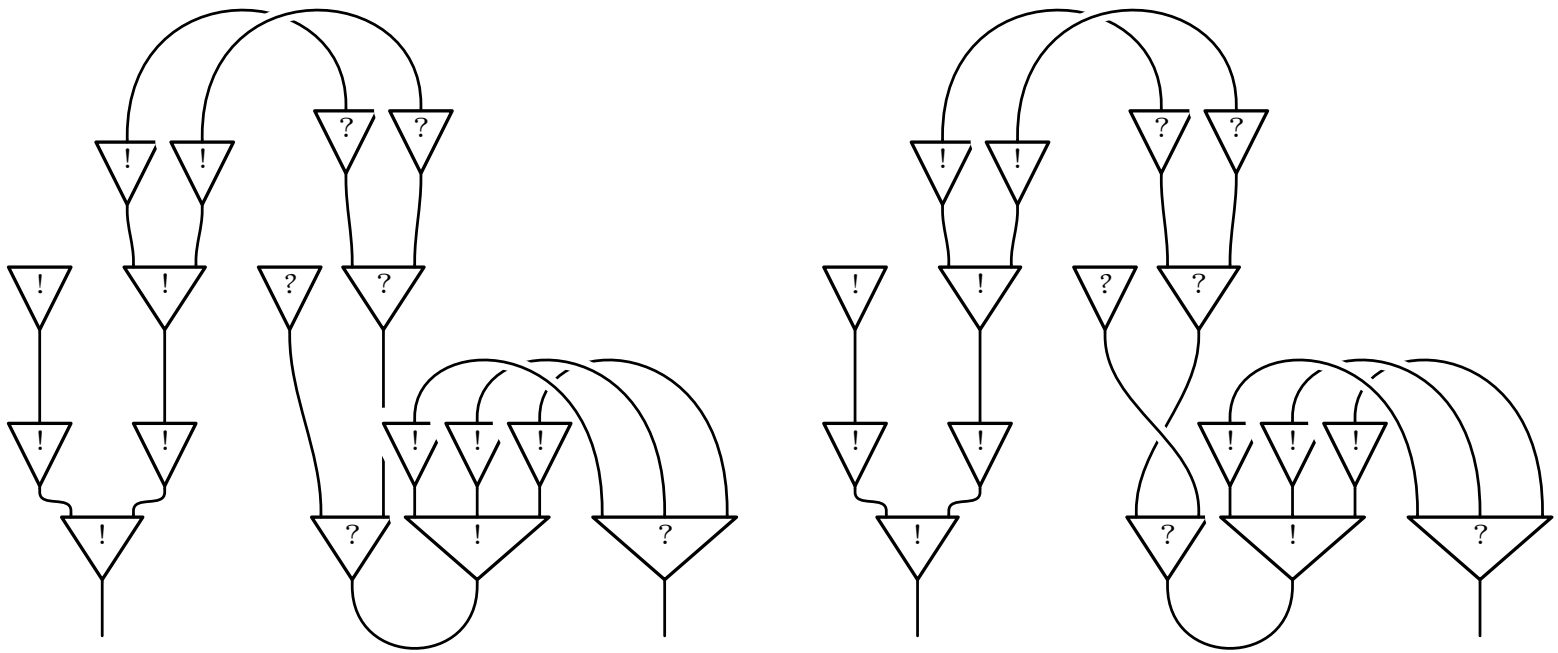


Le groupe d'isotropie

Calcul des coefficients

Soit p un réseau différentiel apparaissant dans le développement de Taylor d'un réseau de la logique linéaire. Le coefficient $c(p)$ est lié au nombre de réseaux isomorphes à p modulo permutation des arbres de co-contraction et des arbres de contraction.

Un exemple : 3!2!2!

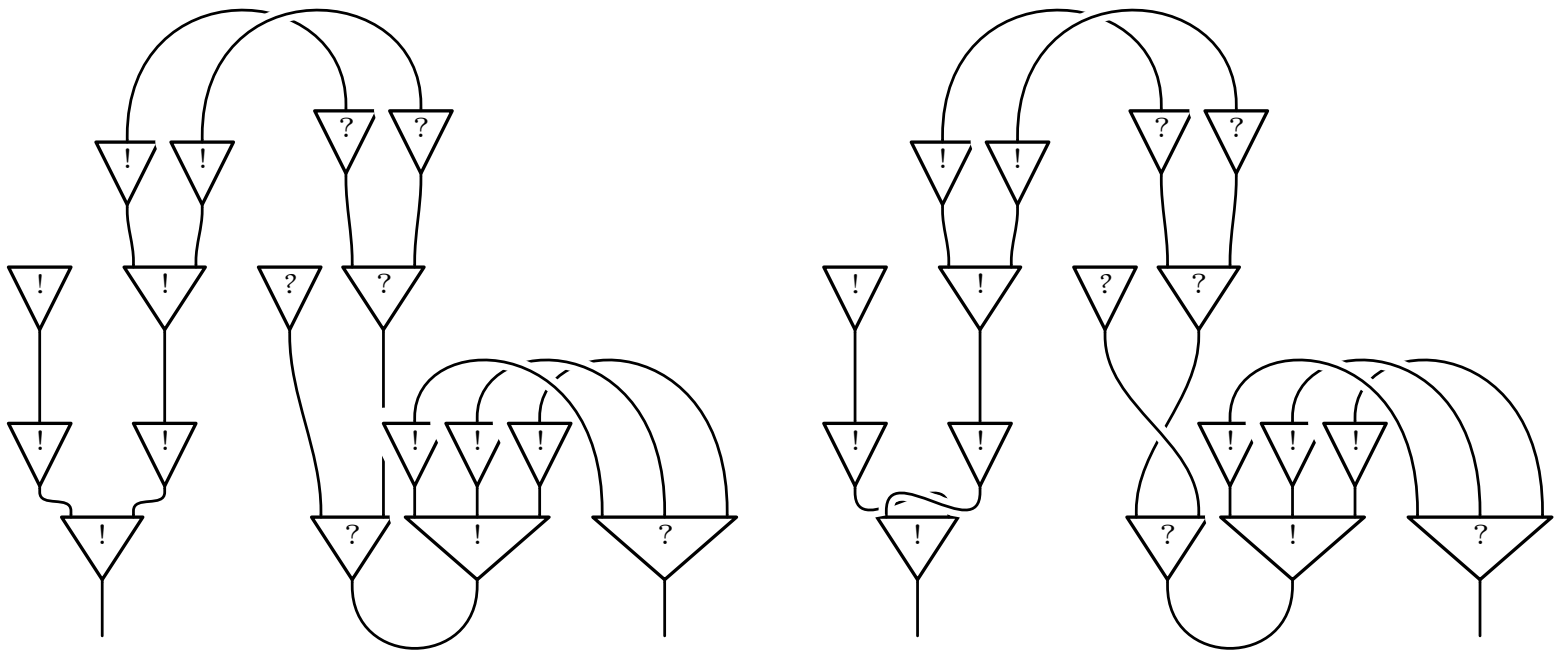


Le groupe d'isotropie

Calcul des coefficients

Soit p un réseau différentiel apparaissant dans le développement de Taylor d'un réseau de la logique linéaire. Le coefficient $c(p)$ est lié au nombre de réseaux isomorphes à p modulo permutation des arbres de co-contraction et des arbres de contraction.

Un exemple : 3!2!2!

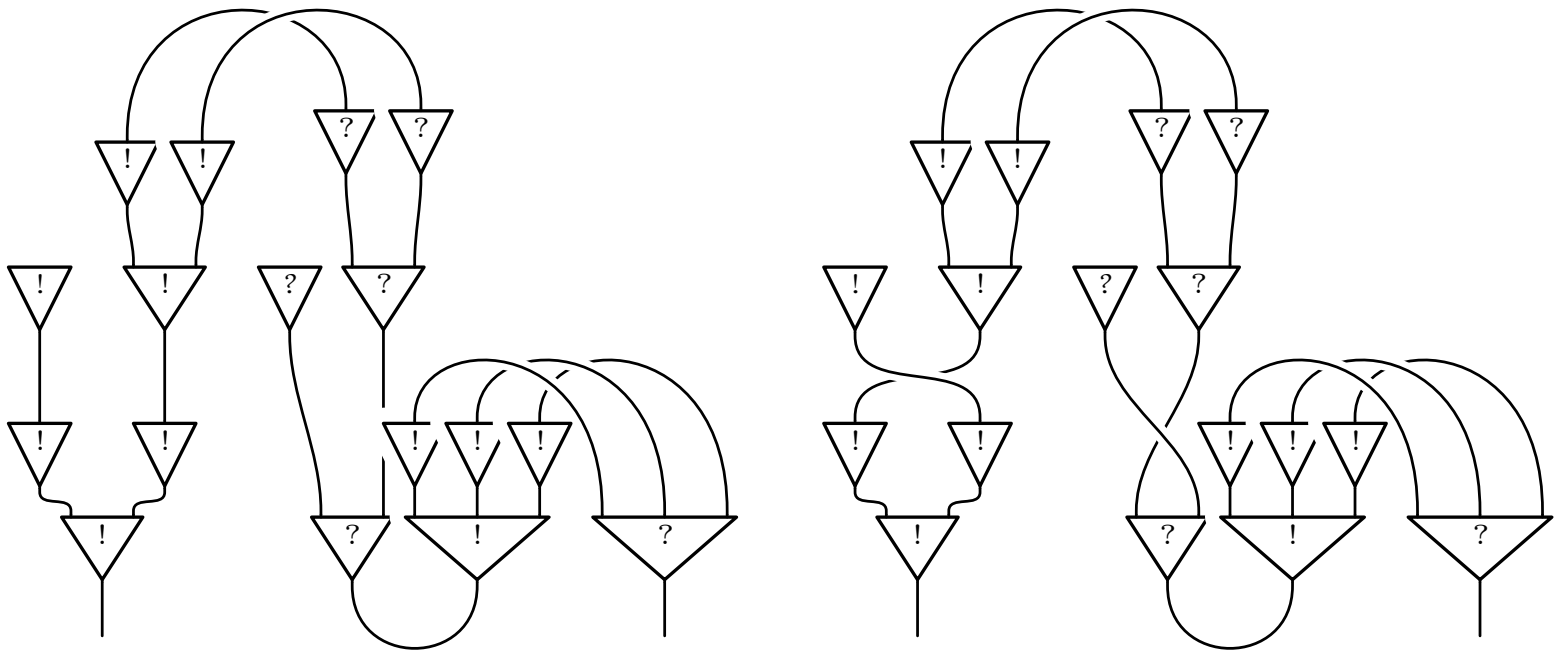


Le groupe d'isotropie

Calcul des coefficients

Soit p un réseau différentiel apparaissant dans le développement de Taylor d'un réseau de la logique linéaire. Le coefficient $c(p)$ est lié au nombre de réseaux isomorphes à p modulo permutation des arbres de co-contraction et des arbres de contraction.

Un exemple : 3!2!2!

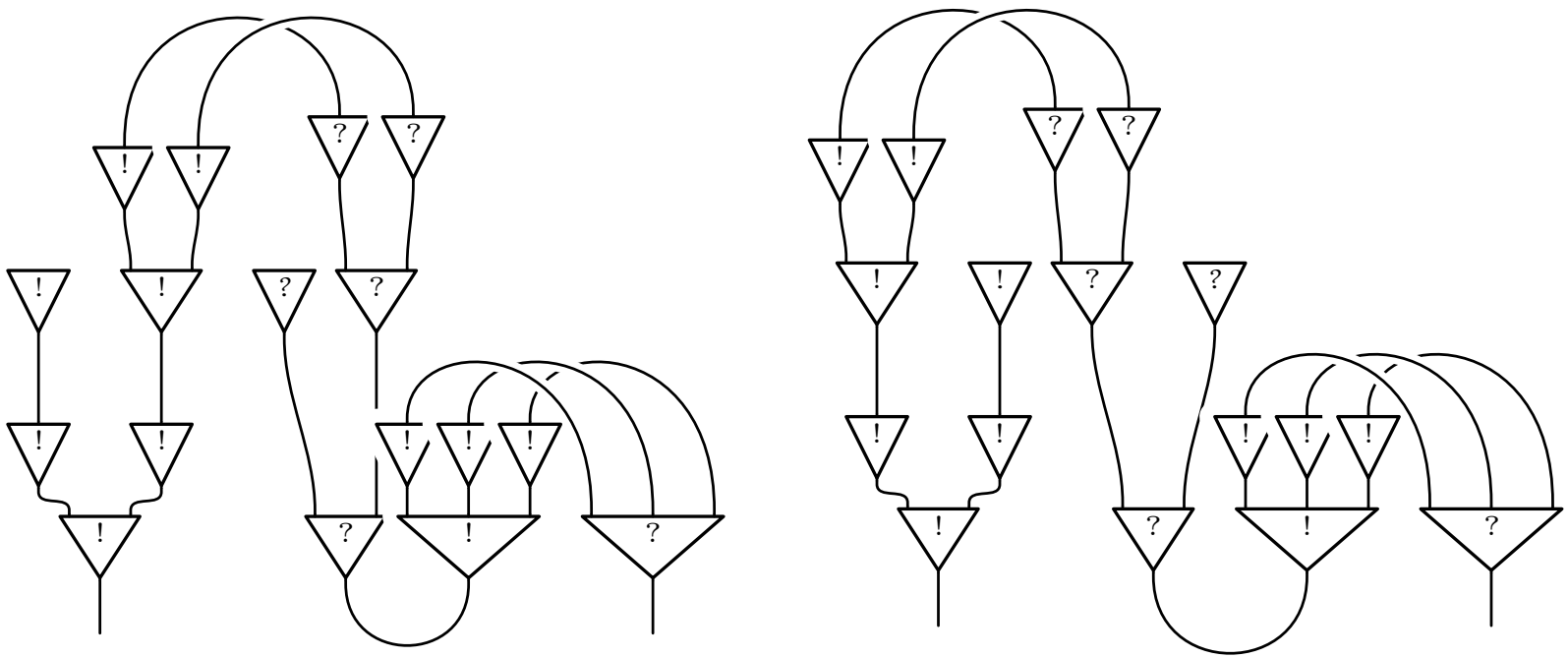


Le groupe d'isotropie

Calcul des coefficients

Soit p un réseau différentiel apparaissant dans le développement de Taylor d'un réseau de la logique linéaire. Le coefficient $c(p)$ est lié au nombre de réseaux isomorphes à p modulo permutation des arbres de co-contraction et des arbres de contraction.

Un exemple : 3!2!2!

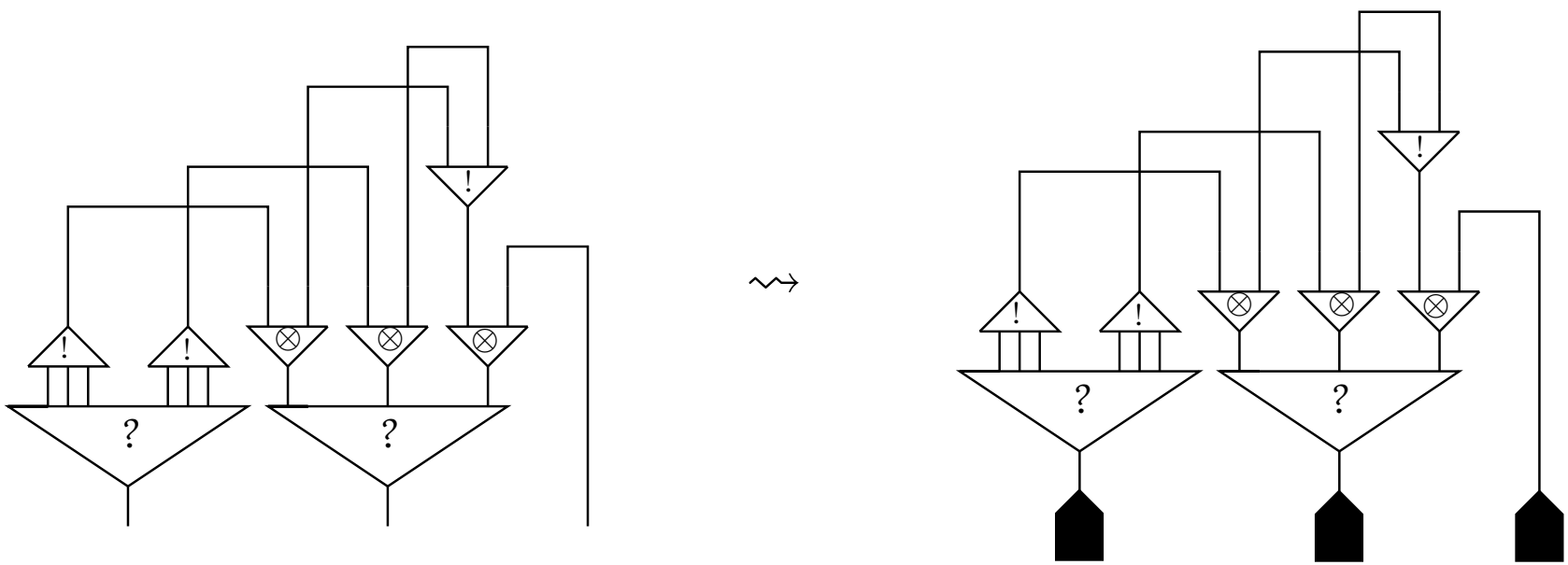


Recoller les réseaux :

Le problème

Etant donné un réseau différentiel, comment savoir s'il apparaît dans le développement de Taylor d'un réseau de la logique linéaire ?

Le recollement, un exemple

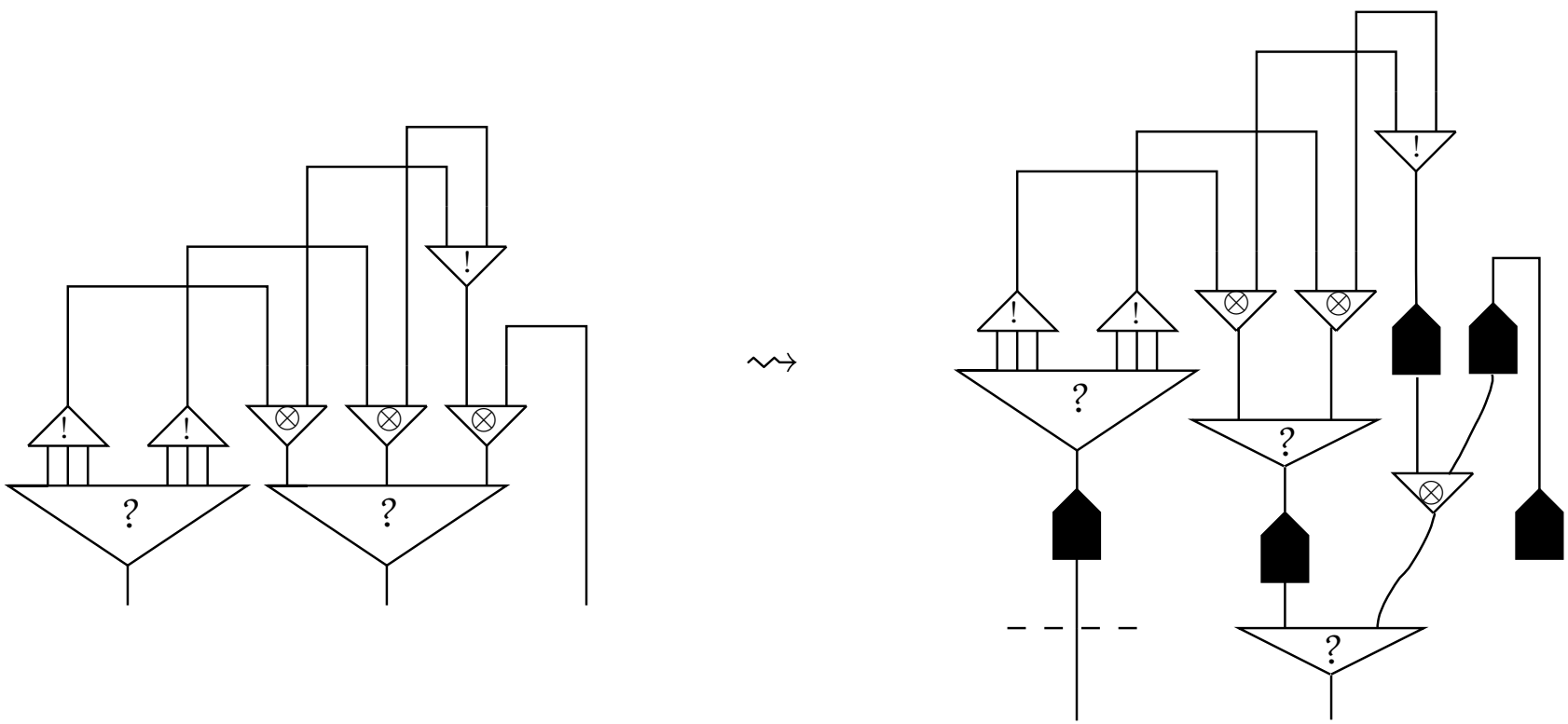


Recoller les réseaux :

Le problème

Etant donné un réseau différentiel, comment savoir s'il apparaît dans le développement de Taylor d'un réseau de la logique linéaire ?

Le recollement, un exemple

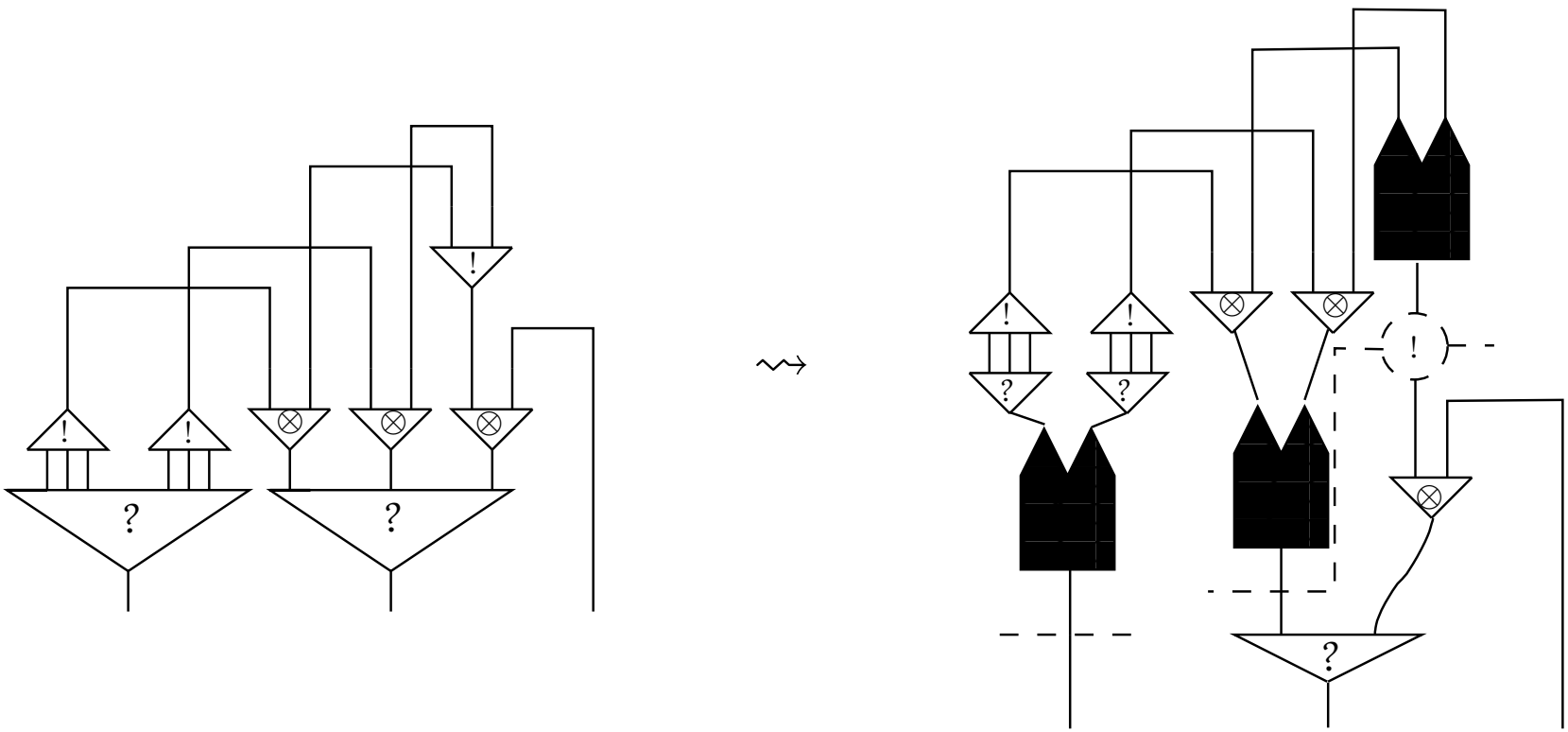


Recoller les réseaux :

Le problème

Etant donné un réseau différentiel, comment savoir s'il apparaît dans le développement de Taylor d'un réseau de la logique linéaire ?

Le recollement, un exemple

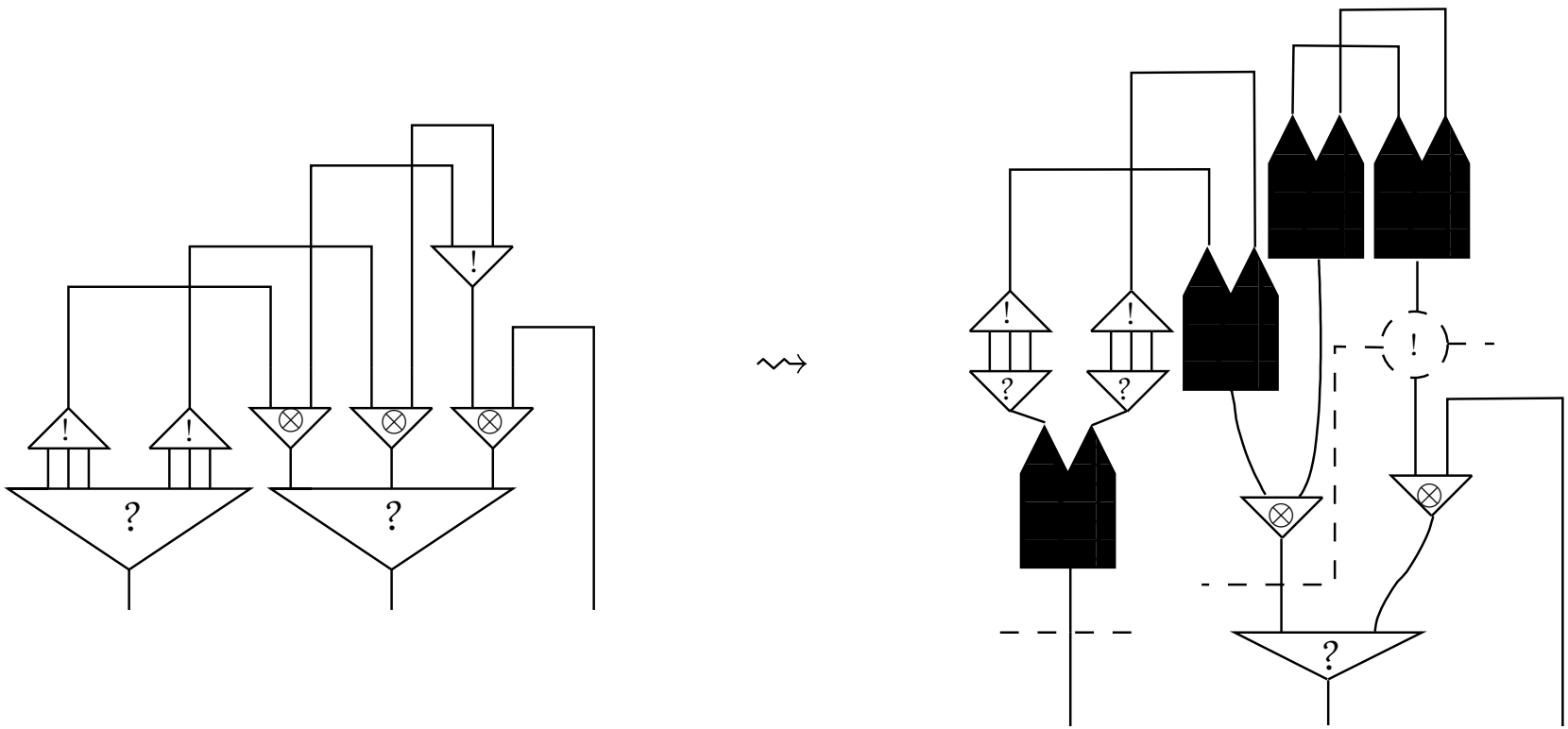


Recoller les réseaux :

Le problème

Etant donné un réseau différentiel, comment savoir s'il apparaît dans le développement de Taylor d'un réseau de la logique linéaire ?

Le recollement, un exemple

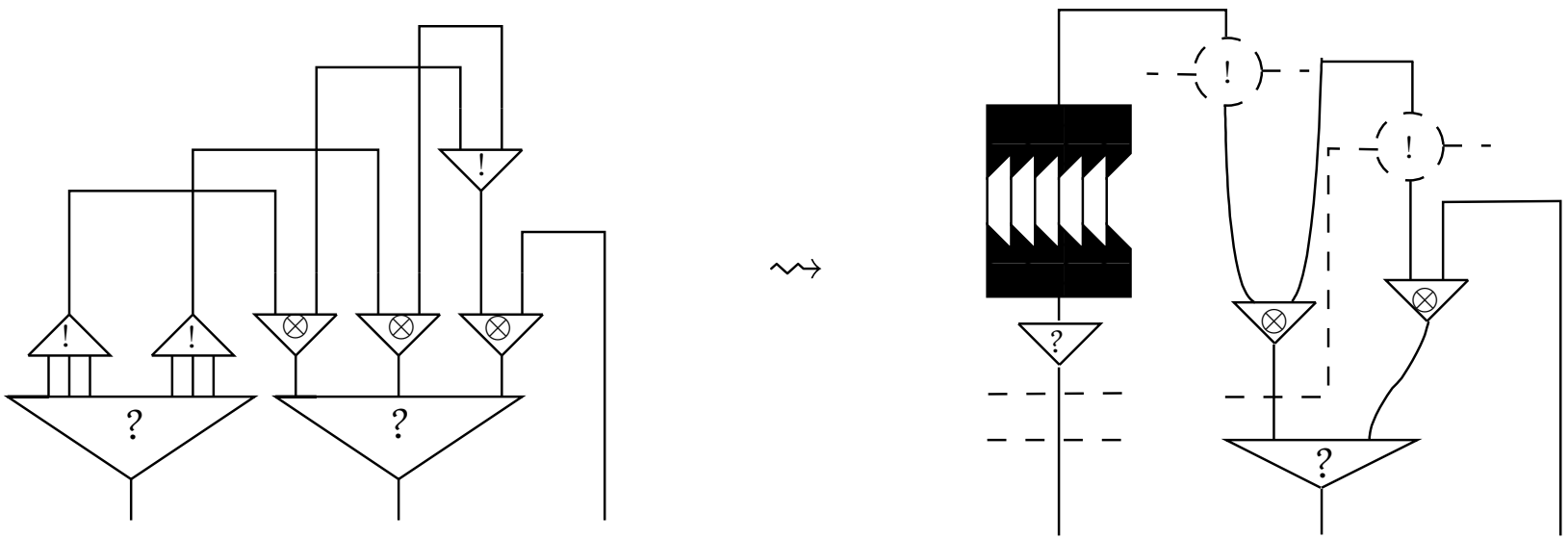


Recoller les réseaux :

Le problème

Etant donné un réseau différentiel, comment savoir s'il apparaît dans le développement de Taylor d'un réseau de la logique linéaire ?

Le recollement, un exemple

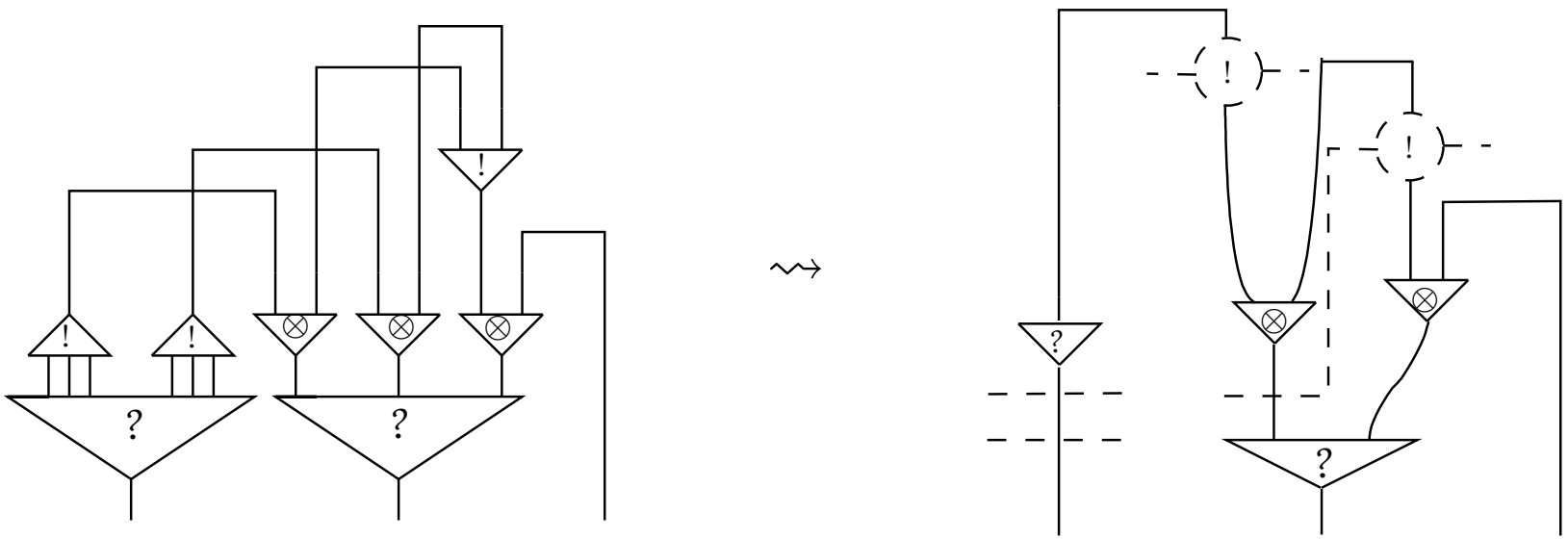


Recoller les réseaux :

Le problème

Etant donné un réseau différentiel, comment savoir s'il apparaît dans le développement de Taylor d'un réseau de la logique linéaire ?

Le recollement, un exemple

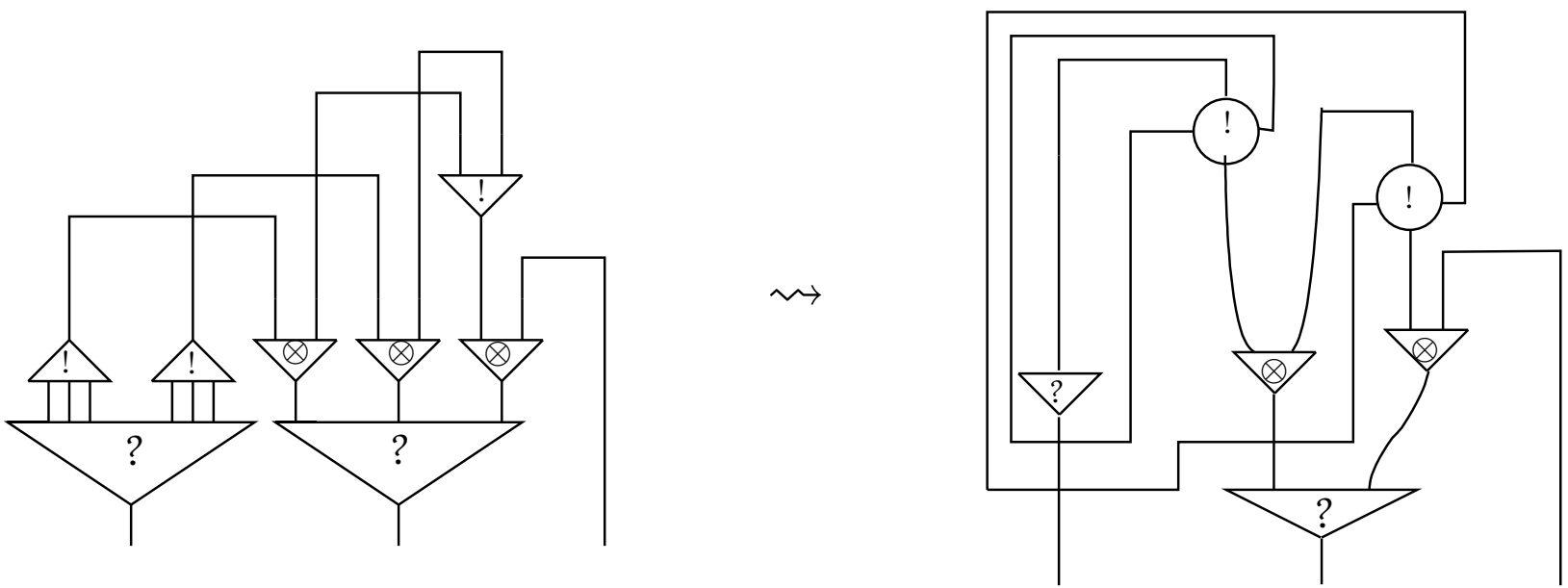


Recoller les réseaux :

Le problème

Etant donné un réseau différentiel, comment savoir s'il apparaît dans le développement de Taylor d'un réseau de la logique linéaire ?

Le recollement, un exemple



Conclusion

Résumé :

- Un modèle des programmes.
- Une analogie : la linéarité, généralisée à la formule de Taylor.
- Création d'un langage de programmation plus atomique et des applications inattendues (la concurrence).
- De nouveaux modèles.

Bibliographie


[Girard] *Linear Logic*, Theoretical Computer Science, 1987


[Ehrhard and Regnier] *The differential λ-calculus*, Theoretical Computer Science, 2003


[Ehrhard and Regnier] *Differential Interaction Nets*, Electronic Notes in Theoretical Computer Science, 2005


[Ehrhard] *Finiteness spaces*, Mathematical Structures in Computer Science, 2005