Composition d'option informatique n°6

(Durée : 4 heures)

L'utilisation de la calculatrice **n'est pas autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Logique linéaire multiplicative

d'après un sujet de Florian Hatat

Dans ce problème, on introduit la syntaxe de la logique dite linéaire multiplicative et on montre quelques résultats.

On considère un ensemble dénombrable de variables, noté $\mathcal{V}$. L'ensemble $\mathcal{F}$ des formules de la **logique linéaire multiplicative** est l'ensemble défini par induction par :

- pour tout $x \in \mathcal{V}$, $x \in \mathcal{F}$;
- si $x \in \mathcal{V}$ alors $x^\perp \in \mathcal{F}$. Le connecteur $\_^\perp$ est appelé **négation linéaire** ;
- si $A_1, A_2 \in \mathcal{F}$ alors $A_1 \otimes A_2 \in \mathcal{F}$. Le connecteur $\otimes$ est appelé **tenseur** ;
- si $A_1, A_2 \in \mathcal{F}$ alors $A_1 \parr A_2 \in \mathcal{F}$. Le connecteur $\parr$ est appelé **par**.

On définit, si A est une formule, la notation $A^\perp$ en posant par induction sur la formule :

$$(x^\perp)^\perp = x \qquad (A \otimes B)^\perp = A^\perp \parr B^\perp \qquad (A \parr B)^\perp = A^\perp \otimes B^\perp$$

Attention, appliquée à une formule, $(\_)^\perp$ n'est qu'une notation, pas un connecteur logique. Il s'agit juste de « sucre syntaxique » destiné à simplifier l'écriture. En particulier, ce n'est pas un cas à prendre en compte lorsqu'on travaille par induction sur les formules. On remarque que l'opération $(\_)^\perp : \mathcal{F} \to \mathcal{F}$ est alors une involution, c'est-à-dire que pour toute formule A, $(A^\perp)^\perp = A$.

En logique linéaire, un **séquent** est donné par un multi-ensemble, c'est-à-dire un ensemble avec multiplicité, de formules :

$$\vdash A_1, A_2, \cdots, A_n$$

Ainsi :

- on s'autorise à échanger l'ordre des formules ; on a donc l'égalité de séquents $\vdash A, B, C = \vdash B, A, C$;
- en revanche, le nombre fois où apparait chaque formule est important. Les deux séquents suivants sont différents : $\vdash A, B, A \neq \vdash A, B$.

Dans un séquent de la logique linéaire, il n'y a que des formules à droite du symbole $\vdash$. Intuitivement, le tenseur se comporte comme une conjonction et le par se comporte comme une disjonction.

Les quatre règles de déduction de la logique linéaire sont les suivantes :

$$\text{pour } x \in \mathcal{V} : \frac{}{\vdash x, x^\perp}\,\text{ax} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\,\text{cut} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, A \otimes B, \Delta}\,\otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr$$

On peut remarquer les faits suivants :

- l'axiome (ax) n'est valable que pour les **variables**, pas pour des formules en général,
- pour la règle du tenseur, la liste des formules des prémisses est une partition des formules du séquent conclusion : aucune formule n'est dupliquée ni supprimée.

1 Préliminaires

1.1 Premières preuves

Question 1 Donner un arbre de preuve de $\vdash ((x \otimes y^\perp) \parr x^\perp) \parr y$.

On définit l'**implication linéaire** (dite **sucette**) $A \multimap B$ comme étant égale à la formule $A^\perp \parr B$. Il s'agit juste d'une notation syntaxique, le connecteur $\multimap$ n'étant pas un constructeur des formules de la logique linéaire.

Exemple

On a l'égalité $(x \otimes (y \parr t^\perp)) \multimap z = (x^\perp \parr (y^\perp \otimes t)) \parr z$.

Question 2 Montrer que les deux séquents suivants sont prouvables :

(a) $\vdash ((x \parr y) \parr z) \multimap (x \parr (y \parr z))$;

(b) $\vdash ((x \multimap y) \otimes (y \multimap z)) \multimap (x \multimap z)$;

où x, y et z sont des variables.

Question 3 La règle axiome n'est valable que pour des variables. Cependant, on voudrait l'utiliser également pour des formules, sous la forme :

$$\frac{}{\vdash A, A^\perp}\ \text{axF}$$

Montrer, par induction structurelle sur la formule A, que cette règle est **admissible**, c'est-à-dire que l'on peut construire un arbre de preuve dont la conclusion est $\vdash A, A^\perp$ en utilisant uniquement les quatre règles de la partie précédente.

Question 4 Démontrer que la règle suivante est dérivable en logique linéaire en incluant la règle (axF) :

$$\frac{\vdash \Gamma, A \multimap B \qquad \vdash \Delta, A}{\vdash \Gamma, \Delta, B}\ \multimap_e$$

Soit A une formule et x une variable. On définit par induction sur la formule le nombre de fois où x apparait dans A, que l'on note $|A|_x$ par (où $y \in \mathcal{V} \setminus \{x\}$) :

$$|x|_x = 1 \qquad |x^\perp|_x = 0 \qquad |y|_x = |y^\perp|_x = 0 \qquad |A \otimes B|_x = |A \parr B|_x = |A|_x + |B|_x$$

On définit de manière similaire le nombre de fois où $x^\perp$ apparait dans A.

Question 5 Montrer que pour toute formule A, $|A^\perp|_x = |A|_{x^\perp}$.

Question 6 En déduire que si A est une formule démontrable en logique linéaire et x une variable, alors $|A|_x = |A|_{x^\perp}$.

Question 7 En déduire que la formule $(x \otimes x) \multimap x$ n'est pas prouvable en logique linéaire.

1.2 Implémentation

On représente une formule de la logique linéaire en OCaml par le type :

```
type formule =
    | Var of int
    | Neg of int
    | Par of formule * formule
    | Tens of formule * formule
```

tel que si l'ensemble des variables est $\{x_0, \ldots, x_{n-1}\}$, alors `Var i` représente x_i, `Neg i` représente $x_i^\perp$, et si A et B sont des formules encodées par `a` et `b`, alors `Par (a, b)` et `Tens (a, b)` représentent $A \parr B$ et $A \otimes B$ respectivement.

Question 8 Écrire une fonction `egales (a: formule) (b: formule) : bool` qui teste l'égalité entre deux formules. Déterminer la complexité de la fonction `egales` en fonction de la taille de ses arguments.

Question 9 Écrire une fonction `neg (a: formule) : formule` qui prend en argument une formule A et renvoie $A^\perp$.

Un séquent est alors une liste de formules.

```
type sequent = formule list
```

Question 10 Écrire une fonction `filtre (a: formule) (seq: sequent) : sequent` qui prend en argument une formule A et un séquent $\vdash \Gamma, A$, et renvoie le séquent $\vdash \Gamma$, c'est-à-dire où on a enlevé **une** occurrence de A. La fonction lèvera l'exception `ErreurPreuve` si la formule A n'apparaît pas dans le séquent donné en argument.

On représente un arbre de preuve avec le type arborescent suivant :

```
type preuve =
    | Ax of int
    | Cut of formule * preuve * preuve
    | PPar of formule * formule * preuve
    | PTens of formule * formule * preuve * preuve
```

de telle sorte que :

- $\dfrac{}{\vdash x_i, x_i^\perp}$ ax est représentée par `Ax i` ;
- si P_1 et P_2 sont des preuves de $\vdash \Gamma, A$ et $\vdash \Delta, A^\perp$ respectivement, alors la preuve $\dfrac{P_1 \qquad P_2}{\vdash \Gamma, \Delta}$ cut est représentée par `Cut (a, p1, p2)` ;
- si P est une preuve de $\vdash \Gamma, A, B$, alors la preuve $\dfrac{P}{\vdash \Gamma, A \parr B}$ $\parr$ est représentée par `PPar (a, b, p)` ;
- si P_1 et P_2 sont des preuves de $\vdash \Gamma, A$ et $\vdash \Delta, B$ respectivement, alors la preuve $\dfrac{P_1 \qquad P_2}{\vdash \Gamma, A \otimes B, \Delta}$ $\otimes$ est représentée par `PTens (a, b, p1, p2)`.

Question 11 Écrire une fonction `verification (p: preuve) : sequent` qui prend en argument une preuve et renvoie le séquent prouvé par cette preuve. Si la preuve est mal formée, la fonction lèvera l'exception `ErreurPreuve`.

Question 12 Justifier que pour une preuve bien formée, la complexité de la fonction `verification` est polynomiale en la taille de la preuve donnée en argument, c'est-à-dire le nombre de règles apparaissant dans la preuve.

On modifiera le code de la fonction `verification` si ce n'est pas le cas.

2 Réseaux de preuve

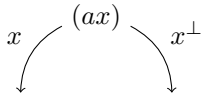
2.1 Réseau

Un réseau est un graphe **orienté** dont certaines arêtes sont pendantes (avec un sommet source, mais pas de destination), dont les sommets sont étiquetés par (ax), (cut), ($\otimes$) ou ($\parr$), et dont les arêtes sont étiquetées par des formules.

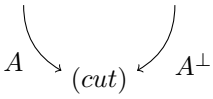
Les arêtes entrantes d'un sommet sont les **prémisses du sommet** et les arêtes sortantes d'un sommet sont les **conclusion du sommet**. On dit qu'un sommet t est **voisin** d'un sommet s s'il y a une arête de s vers t.

Un réseau doit, de plus, vérifier les conditions suivantes :

- un sommet (ax) n'a pas de prémisse et deux conclusions, étiquetées par x et $x^\perp$ respectivement, où $x \in \mathcal{V}$;



- un sommet (cut) n'a pas de conclusion et deux prémisses, étiquetées par A et $A^\perp$, où $A \in \mathcal{F}$ respectivement ;



- un sommet ($\otimes$) (resp. ($\parr$)) a deux prémisses étiquetées par A et B respectivement, où $A, B \in \mathcal{F}$, et une conclusion étiquetée par $A \otimes B$ (resp. $A \parr B$).

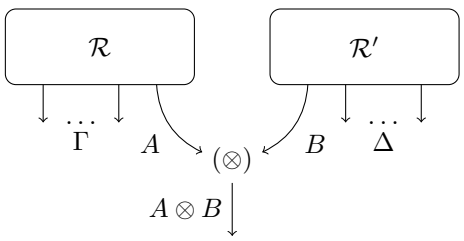
Les étiquettes des arêtes pendantes d'un réseau sont appelées les **conclusions du réseau**.

Question 13 Montrer qu'un réseau ne possède pas de cycle orienté.

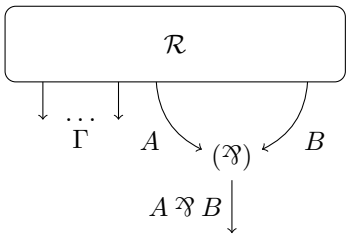
2.2 Réseaux de preuve

À chaque arbre de preuve de $\vdash A_1, \cdots, A_n$, on associe par induction un réseau dont les conclusions sont $A_1, \cdots, A_n$:

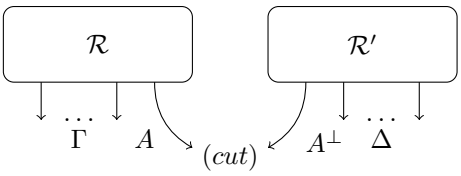
- la règle axiome donne un sommet (ax) avec les mêmes conclusions ;
- si on a un réseau $\mathcal{R}$ de conclusions Γ, A et un réseau $\mathcal{R}'$ de conclusions Δ, B, alors on peut construire un réseau de conclusions $\Gamma, A \otimes B, \Delta$ en reliant la conclusion A de $\mathcal{R}$ et la conclusion B de $\mathcal{R}'$ à un nouveau sommet ($\otimes$) ;



– pour la règle du par, on procède de même en reliant à un nouveau sommet ($\parr$) les conclusions A et B d'un réseau de conclusions Γ, A, B ;



– pour la règle de la coupure, on procède comme pour ($\otimes$).



Définition

Un **réseau de preuve** est un réseau obtenu de cette façon à partir d'un arbre de preuve.

La figure 1 donne un exemple de réseau de preuve.

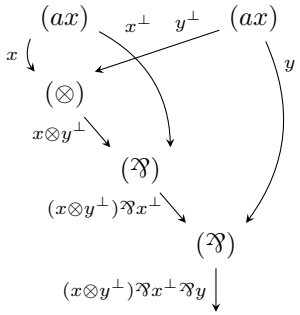


FIGURE 1 – Un réseau de preuve de $\vdash (x \otimes y^\perp) \parr x^\perp \parr y$.

Question 14 Donner un réseau de preuve de $\vdash (((x \parr y) \otimes x^\perp) \parr (x^\perp \otimes y^\perp)) \parr x$. On se contentera de n'indiquer les étiquettes des arêtes que sur les conclusions des sommets (ax).

Question 15 Montrer qu'il existe un réseau qui n'est pas un réseau de preuve.

2.3 Critère de Danos et Régnier

Définition : Interrupteur

Un **interrupteur** d'un réseau $\mathcal{R}$ (pas nécessairement de preuve) est un graphe non orienté obtenu à partir de $\mathcal{R}$ en :

- supprimant toutes les arêtes pendantes ;
- supprimant une des prémisses de chaque nœud ($\parr$) ;
- désorientant toutes les arêtes restantes.

Question 16 Dessiner un exemple d'interrupteur du réseau de preuve de la figure 1.

Question 17 Déterminer le nombre d'interrupteurs d'un réseau possédant k nœuds ($\parr$).

Définition 2.1

On dit qu'un réseau vérifie le **critère de Danos et Régnier** si tous ses interrupteurs sont des arbres (au sens graphe connexe sans cycle).

Pour la partie suivante, on utilisera le théorème suivant :

Théorème : Critère de correction de Danos et Régnier

Un réseau sans sommet (cut) est un réseau de preuve si et seulement s'il vérifie le critère de Danos et Régnier.

Question 18 En procédant par induction sur les arbres de preuve, démontrer le sens direct du théorème : si $\mathcal{R}$ est un réseau de preuve alors il vérifie le critère de Danos et Régnier.

On admettra le sens réciproque du théorème.

2.4 Un théorème d'élimination des coupures

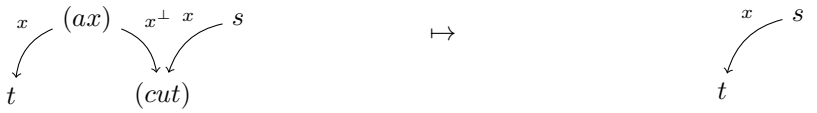
On considère un réseau de preuve obtenu à partir d'une preuve qui utilise éventuellement la règle (cut).

Question 19 Montrer qu'un tel réseau vérifie le critère de Danos et Régnier.

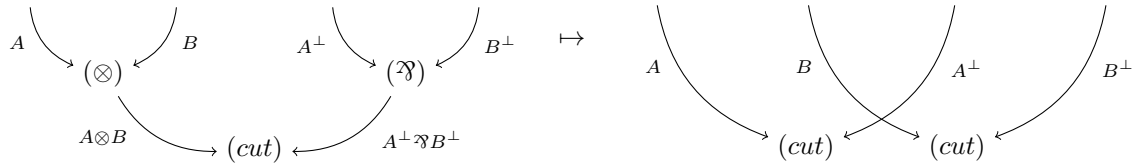
Question 20 Montrer qu'un sommet (cut) est voisin d'un sommet $(\otimes)$ si et seulement s'il est voisin d'un sommet $(\parr)$.

Sur un réseau, on définit une étape d'élimination d'une coupure de la façon suivante :

– si un sommet (cut) est voisin d'un sommet (ax), peu importe de quel côté, on supprime ces deux sommets :



– si un sommet (cut) est voisin d'un sommet $(\otimes)$ et d'un sommet $(\parr)$, on le découpe en deux sommets (cut) de la façon suivante :



L'algorithme d'élimination des coupures consiste à répéter l'étape d'élimination d'une coupure tant que c'est possible.

Question 21 Montrer que si le réseau vérifie le critère de Danos et Régnier initialement alors le réseau obtenu à la fin de l'élimination des coupures vérifie le critère de Danos et Régnier.

Question 22 Montrer que l'algorithme d'élimination des coupures termine et que le réseau obtenu n'a plus de sommet (*cut*).

Question 23 En déduire que la règle (cut) est admissible dans le système de déduction formé des règles (ax), ($\parr$) et ($\otimes$).

3 Test de connexité

On suppose que les entêtes `stdlib.h` et `stdbool.h` ont été chargées.

On implémente un graphe non orienté en C par le type suivant :

```
struct Graphe {
    int n;
    int* deg;
    int** adj;
};

typedef struct Graphe graphe;
```

de telle sorte que pour un graphe $G = (\mathcal{S} = [\![0, n-1]\!], \mathcal{A})$ encodé par `g` de type `graphe`, alors `g.n` $= n$, `g.deg` et `g.adj` sont des tableaux de taille n tels que pour $s \in \mathcal{S}$, `g.deg[s]` est le degré de s, c'est-à-dire son nombre de voisins, et `g.adj[s]` est un tableau de taille `g.deg[s]` contenant les voisins de s dans un ordre quelconque.

Pour décider si un réseau est un réseau de preuve, le critère de Danos et Régnier permet de se ramener à des tests sur les interrupteurs.

Question 24 Écrire une fonction `bool connexe(graphe g)` qui teste si un graphe est connexe ou non.

Question 25 En déduire une fonction `bool arbre(graphe g)` qui teste si un graphe est un arbre ou non.

Question 26 Déterminer la complexité temporelle **ET** spatiale de la fonction `arbre` en fonction de $|\mathcal{S}|$ et $|\mathcal{A}|$.

Pour améliorer la complexité spatiale de la fonction `connexe`, on propose d'utiliser un algorithme probabiliste. Une **marche aléatoire** dans un graphe $G = (\mathcal{S}, \mathcal{A})$ depuis un sommet $s \in \mathcal{S}$ est un algorithme qui part de s et, à chaque étape, choisit de manière aléatoire et uniforme un nouveau sommet parmi les voisins de s. Ce nouveau sommet devient le sommet courant pour l'étape suivante de la marche aléatoire.

On suppose disposer d'une fonction `int randint(int N)` qui renvoie un entier choisi aléatoirement et uniformément entre 0 et $N-1$.

Question 27 Écrire une fonction `bool chemin_alea(graphe g, int s, int t, int etapes)` qui prend en argument un graphe G, un sommet source s et un sommet destination t et effectue une marche aléatoire depuis le sommet s et renvoie `true` si la marche a permis d'atteindre le sommet t en moins de `etapes` itérations, et `false` sinon.

Cette fonction devra avoir une complexité spatiale constante.

Dans un graphe non orienté connexe $G = (\mathcal{S}, \mathcal{A})$, pour $s, t \in \mathcal{S}$, on note $X_{s,t}$ la variable aléatoire qui donne le nombre d'étapes réalisées par une marche aléatoire de s à t.

Question 28 Justifier que $\mathbb{E}(X_{s,t}) = \sum_{u \text{ voisin de } s} \frac{1}{\deg(s)}(1+\mathbb{E}(X_{u,t}))$, où $\deg(s)$ est le nombre de voisins de s.

On admet qu'en faisant un parallèle entre le graphe et un circuit électrique, on peut utiliser la loi des nœuds et la loi des mailles pour montrer que $\mathbb{E}(X_{s,t}) \leqslant |\mathcal{S}|^3$.

Question 29 En déduire, en justifiant, une fonction `bool connectes_alea(graphe g, int s, int t)` correspondant à un algorithme de type Monte Carlo, de complexité temporelle polynomiale et de complexité spatiale constante, qui détermine si s et t sont dans la même composante connexe de G et tel que :

- l'algorithme n'a pas de faux positif ;
- la probabilité de faux négatif est inférieure ou égale à $\frac{1}{2}$.

4 Logique directe

On considère le problème de décision suivant : `Logique Linéaire Multiplicative (LLM)` :

* **Instance** : un séquent $\vdash \Gamma$.
* **Question** : existe-t-il une preuve de $\vdash \Gamma$ en logique linéaire multiplicative ?

Question 30 Montrer que $\texttt{LLM} \in \mathsf{NP}$. On pourra utiliser le théorème d'élimination des coupures.

On considère la règle de déduction suivante, appelée **affaiblissement** :

$$\frac{\vdash \Gamma}{\vdash \Gamma, \Delta}\ \text{aff}$$

On appelle **logique directe** le système de déduction formé des règles de la logique linéaire multiplicative et de l'affaiblissement.

Pour la suite, on s'intéresse au problème de décision `Logique Directe (LD)` :

* **Instance** : un séquent $\vdash \Gamma$.
* **Question** : existe-t-il une preuve de $\vdash \Gamma$ en logique directe ?

On admet qu'un théorème d'élimination des coupures en logique directe permet de montrer de manière similaire à la question précédente que $\texttt{LD} \in \mathsf{NP}$.

On souhaite montrer que LD est NP-complet par une réduction depuis le problème suivant, qu'on admet comme étant NP-complet : `Couverture des arêtes par les sommets (CPS)` :

* **Instance** : un graphe non orienté $G = (\mathcal{S}, \mathcal{A})$ et un entier k.
* **Question** : existe-t-il une couverture des arêtes par les sommets de G, c'est-à-dire un ensemble de sommets qui intersecte chaque arête, de cardinal $\leqslant k$?

Pour ce faire, on considère un graphe $G = (\mathcal{S}, \mathcal{A})$ et un entier k. Pour simplifier, on suppose que G n'a pas de sommet isolé (de degré 0).

On pose $\mathcal{V} = \mathcal{S} \cup \{x\}$. Pour $A \in \mathcal{F}$, et $m \in \mathbb{N}^*$, on note A^m la formule $\underbrace{A \parr A \parr \dots \parr A}_{m \text{ fois}}$ (par abus de notation, on fait comme si $\parr$ est associatif).

On définit les formules suivantes :

- $A(G) = \bigparr_{s \in \mathcal{S}} \left(x^\perp \otimes (s^\perp)^{\deg(s)}\right)$;
- $B(G) = \bigotimes_{\{s,t\} \in \mathcal{A}} (s \parr t)$.

On pose alors $\vdash \Gamma(G,k)$ le séquent $\vdash x^k, A(G), B(G)$.

Question 31 Déterminer le séquent $\vdash \Gamma(G_0, 2)$ pour G_0 le graphe représenté figure 2.

FIGURE 2 – Le graphe G_0.

On veut montrer que G possède une couverture de cardinal k si et seulement si $\vdash \Gamma(G,k)$ est prouvable en logique directe.

Pour les deux questions suivantes, on suppose que X est une couverture des arêtes de G de cardinal k. On pose $\vdash \Delta$ le séquent contenant, pour chaque $s \in X$, la formule $s^\perp$ apparaissant autant de fois que son degré. Par exemple, pour $X = \{s_1, s_3\}$ dans le graphe G_0, on aurait $\vdash \Delta$ égal au séquent $\vdash s_1^\perp, s_1^\perp, s_3^\perp$.

Question 32 Montrer que $\vdash \Delta, B(G)$ est prouvable en logique directe.

Question 33 En déduire que $\vdash \Gamma(G,k)$ est prouvable en logique directe.

Dans une preuve d'un séquent, on dit qu'une formule A est **affaiblie** s'il existe une application de la règle $\dfrac{\vdash \Gamma}{\vdash \Gamma, \Delta}$ aff, avec $A \in \Delta$.

Question 34 Soit $\vdash \Gamma$ un séquent prouvable en logique directe. Montrer qu'il existe un arbre de preuve de $\vdash \Gamma$ vérifiant les conditions suivantes :

(a) pour chaque occurrence d'une formule $A \otimes B$, ni A, ni B n'est affaiblie ;

(b) pour chaque occurrence d'une formule $A \parr B$, au moins l'une des deux parmi A et B n'est pas affaiblie.

Attention, lorsqu'on parle de A ou B pour cette question, on ne parle que de **l'occurrence** de A ou B correspondant à l'application de la règle $\otimes$ ou $\parr$.

On suppose que $\vdash \Gamma(G,k)$ est prouvable en logique directe. On pose P une preuve de $\vdash \Gamma(G,k)$. On pose également X l'ensemble des sommets $s \in \mathcal{S}$ tels que l'application de la règle $\dfrac{}{\vdash s, s^\perp}$ ax apparait dans P.

Question 35 On suppose dans cette question que X n'est pas une couverture des arêtes par les sommets. Montrer qu'il existe une preuve de $\vdash \Gamma(G,k)$ où $B(G)$ est affaiblie. En déduire une contradiction.

Question 36 On suppose dans cette question que $|X| > k$. Montrer qu'il existe une preuve de $\vdash \Gamma(G,k)$ où au moins $|X|$ sous-formules de la forme $x^\perp \otimes (s^\perp)^{\deg(s)}$ qui ne sont pas affaiblies. En déduire une contradiction.

Question 37 En déduire que LD est NP-difficile.

⋆ ⋆ ⋆