Composition d'Informatique n°2

Sujet 2 : Transducteurs finis

Proposition de corrigé

⋆ ⋆ ⋆

Remarques fréquentes

- 4A. Attention, les écritures binaires sont inversées, bit de poids faible à gauche.
- 9A. De l'égalité $\Phi(\varepsilon)\Phi(\varepsilon) = \Phi(\varepsilon)$, il faut un argument (par exemple sur la taille) pour justifier que $\Phi(\varepsilon) = \varepsilon$.
- 20A. Attention, l'automate B est non déterministe. En particulier, $\rho(\Delta(q_0, u))$ n'a aucun sens.
- 28A. Il faut se ramener à un transducteur dont tous les états sont accessibles, ce qui ne fait pas partie des hypothèses de cette question.

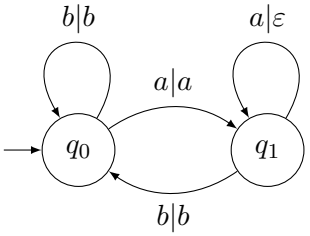
1 Transducteurs séquentiels

1.1 Premiers exemples

Question 1 On a $\varphi_T(bbbaaa) = 01010$ et $\varphi_T(bbab) = 01101$ par exemple.

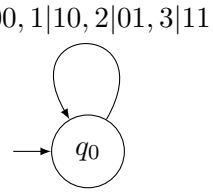
Question 2 Pour un mot $u \in \{a, b\}^*$, $\varphi_T(u)$ est construit en remplaçant chaque séquence de k b consécutifs par $3\left\lfloor \frac{k}{2} \right\rfloor$ b, sans toucher aux séquences de a consécutifs. Par exemple, $\varphi_{T_1}(bbaaabbbbba) = bbbaaabbbbbba$.

Question 3 Le transducteur suivant convient :

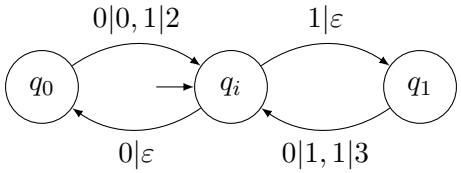


Lorsqu'on se trouve dans l'état q_1, c'est qu'on est en cours de lecture d'une séquence de a consécutifs. À part le premier a qui a été écrit en sortie, tous les suivants doivent être supprimés (c'est ce qui est manifesté par la transition $q_1 \xrightarrow{a|\varepsilon} q_1$). Lorsqu'on lit un b, on arrive forcément en q_0, en écrivant ce b en sortie.

Question 4 On remarque qu'il suffit de remplacer chaque 0 par 00, 1 par 01, 2 par 10 et 3 par 11.



Question 5 On lit les bits deux par deux. On utilise pour cela deux états en plus de l'état initial, indiquant si le premier bit de chaque paire est un 0 ou un 1. On utilise la transformation réciproque de celle décrite ci-dessus.



On appelle ici q_i l'état initial. L'état q_0 indique que la dernière lettre lue est un 0. L'état q_1 indique que la dernière lettre lue est un 1.

1.2 Implémentation

Question 6 On utilise ici une fonction récursive auxiliaire `delta_lamdba : etat -> mot -> etat * mot` telle que pour un état `q` et un mot `u`, `delta_lambda q u` renvoie le couple $(\delta(q,u), \lambda(q,u))$. Pour ce faire, on calcule l'image de q par la première lettre de u, et on fait un appel récursif pour reconstruire la solution. La fonction `calcul` renvoie simplement la valeur $\lambda(q_0, u)$, q_0 étant modélisé par l'état `0`.

```
let calcul t u =
    let rec delta_lambda q = function
        | []     -> q, []
        | a :: u ->
            let q', v = t.(q).(a) in
            let qf, vf = delta_lambda q' u in
            qf, v @ vf
    in
    snd (delta_lambda 0 u)
```

Pour un mot de taille n, cette fonction fait n appels récursifs à `delta_lambda`. Chacun de ces appels fait des opérations en temps constant, et une concaténation. La complexité de cette concaténation est linéaire en la taille du premier mot, c'est-à-dire linéaire en $\lambda(q,a)$. À T fixé, cette concaténation est bien en temps constant (car il existe un nombre fini de transitions).

Question 7 On utilise ici deux fonctions auxiliaires, pour `lst` une liste de mots et `a` une lettre :

- `ajout_lettre lst a` renvoie l'image de `lst` dans laquelle on a ajouté `a` au début de chaque mot ;
- `ajouts_lettres lst a` renvoie une liste de mots dans laquelle on a ajouté en début des mots de `lst` toutes les lettres possibles entre `a` et $p-1$.

Il suffit alors de rajouter toutes les lettres possibles devant les mots de taille $m-1$ pour obtenir les mots de taille m. On remarque que la liste obtenue donne les mots par ordre lexicographique.

```
let rec liste_mots p m =
    let ajout_lettre lst a =
        List.map (fun u -> a :: u) lst
    in
    let rec ajouts_lettres lst a =
        if a = p then []
        else ajout_lettre lst a @ ajouts_lettres lst (a + 1)
    in
    if m = 0 then [[]]
    else ajouts_lettres (liste_mots p (m - 1)) 0
```

Question 8 Comme on n'impose aucune complexité, on fait une fonction naïve qui utilise la fonction précédente. L'idée est, pour chaque valeur $k \in [\![0, m]\!]$, de tester s'il existe un antécédent u de v, de taille k. La fonction `trouver_antecedent` fait cette recherche d'antécédent parmi les mots de taille k, et augmente la taille des mots à considérer si aucun antécédent n'a été trouvé.

```
let antecedent t m v =
    let p = Array.length t.(0) in
    let rec trouver_antecedent k =
        if k = m + 1 then None
        else begin
            let lst = liste_mots p k in
            match List.find_opt (fun u -> calcul t u = v) lst with
                | None -> trouver_antecedent (k + 1)
                | Some u -> Some u
        end
    in
    trouver_antecedent 0
```

1.3 Morphismes de mots

Question 9 En remarquant que $\varepsilon = \varepsilon\varepsilon$, on a $\Phi(\varepsilon) = \Phi(\varepsilon)\Phi(\varepsilon)$. On en déduit que $|\Phi(\varepsilon)| = 2|\Phi(\varepsilon)|$, puis que $\Phi(\varepsilon) = \varepsilon$.

Question 10 On fait une induction sur un mot $u \in \Sigma^*$ pour montrer que $\Phi(u) = \Psi(u)$:

- si $u = \varepsilon$, alors d'après la question précédente, $\Phi(u) = \varepsilon = \Psi(u)$;
- supposons le résultat vrai pour u et soit $a \in \Sigma$. Alors :

$$\Phi(ua) = \Phi(u)\Phi(a) = \Psi(u)\Psi(a) = \Psi(ua)$$

l'égalité du milieu étant par hypothèse sur Φ et Ψ et par hypothèse d'induction, et les deux autres par définition d'un morphisme de mots.

On conclut par induction, et on en déduit que $\Phi = \Psi$.

Question 11 On considère le transducteur $T_\Phi = (\{q_0\}, \Sigma, \Gamma, \delta, \lambda, q_0)$, tel que :

– pour tout $a \in \Sigma$, $\delta(q_0, a) = q_0$;
– pour tout $a \in \Sigma$, $\lambda(q_0, a) = \Phi(a)$.

Une récurrence permet de montrer que pour tout mot $u \in \Sigma^*$, $\varphi_{T_\Phi}(u) = \lambda^*(q_0, u) = \Phi(u)$. On en déduit que Φ est séquentielle.

Question 12 Montrons que φ_T est un morphisme de mots si et seulement si $\forall q, a \in Q \times \Sigma, \lambda(q, a) = \lambda(q_0, a)$.

($\Leftarrow$) Supposons $\forall q, a \in Q \times \Sigma, \lambda(q, a) = \lambda(q_0, a)$. Alors une induction rapide sur v permet de montrer que pour tous $u, v \in \Sigma^*$, $\lambda(\delta(q_0, u), v) = \lambda(q_0, v)$. Dès lors, on a

$$\begin{aligned} \varphi_T(uv) &= \lambda(q_0, u)\lambda(\delta(q_0, u), v) \\ &= \lambda(q_0, u)\lambda(q_0, v) \\ &= \varphi_T(u)\varphi_T(v) \end{aligned}$$

On en déduit que φ_T est bien un morphisme de mots.

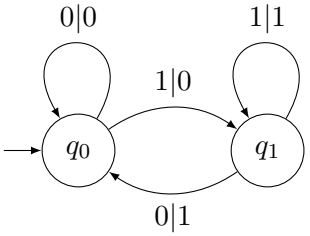
($\Rightarrow$) Raisonnons par contraposée et supposons qu'il existe $q, a \in Q \times \Sigma$ tels que $\lambda(q, a) \neq \lambda(q_0, a)$. Soit $u \in \Sigma^*$ tel que $\delta(q_0, u) = q$ (u existe car tous les états sont supposés accessibles). Alors :

$$\begin{aligned} \varphi_T(ua) &= \lambda(q_0, ua) \\ &= \lambda(q_0, u)\lambda(q, a) \\ &\neq \varphi_T(u)\varphi_T(a) \end{aligned}$$

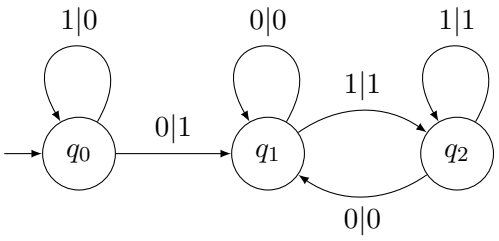
On en déduit que φ_T n'est pas un morphisme.

1.4 Machines de Mealy et de Moore

Question 13 On se contente de rajouter un 0 à la fin du mot. Pour ce faire, on rajoute un 0 au début du mot, et on réécrit chaque bit lu en sortie. Comme l'écriture se fait modulo $2^{|u|}$, le dernier bit ne sera pas écrit en sortie. Les deux états permettent de savoir quel est le dernier bit à avoir été lu.



Question 14 Il faut simuler une propagation de retenue : on continue à propager tant qu'on lit un 1. On ne propage plus dès qu'on lit un 0. On utilise un troisième état pour que la machine soit de Moore.



Question 15 Il suffit de vérifier que toutes les images sont de taille 1.

```
exception PasMealy

let est_mealy t =
    let n = Array.length t and p = Array.length t.(0) in
    try
        for i = 0 to n - 1 do
            for j = 0 to p - 1 do
                match snd t.(i).(j) with
                    | [_] -> ()
                    | _ -> raise PasMealy
            done
        done;
        true
    with _ -> false
```

Dans le pire cas, la machine est de Mealy. On vérifie alors toutes les transitions. Chaque vérification se fait en temps constant (ça ne serait pas le cas si on avait utilisé `List.length`, par exemple). La complexité totale est en $\mathcal{O}(n \times p)$, où $n = |Q|$ et $p = |\Sigma|$.

Question 16 Soit $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_0)$ une machine de Mealy. Posons $M' = (Q \times \Gamma, \Sigma, \Gamma, \delta', \lambda', q_0 \times b_0)$, où $b_0 \in \Gamma$ est une lettre quelconque, en définissant pour $q \in Q$, $b \in \Gamma$ et $a \in \Sigma$:

- $\lambda'((q,b),a) = \lambda(q,a)$;
- $\delta'((q,b),a) = (\delta(q,a), \lambda(q,a))$.

Montrons que M' est une machine de Moore telle que $\varphi_M = \varphi_{M'}$.

- M' est une machine de Moore. En effet, pour une transition $(q_1, b_1) \xrightarrow{a|b} (q_2, b_2)$, on a nécessairement $b = b_2$. De plus, M' est bien une machine de Mealy, car $|\lambda'((q,b),a)| = |\lambda(q,a)| = 1$.
- par induction sur $u = a_1 \ldots a_n \in \Sigma^*$, on peut montrer que :

$$\begin{aligned} \lambda'((q_0,b_0),u) &= \lambda(q_0,u) \\ \delta'((q_0,b_0),u) &= (\delta(q_0,u), \lambda(\delta(q_0,a_1\ldots a_{n-1}),a_n)) \end{aligned}$$

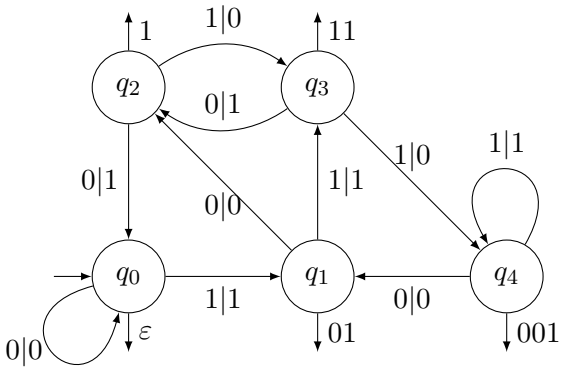
  La relation sur les fonctions de sortie montre bien que $\varphi_M = \varphi_{M'}$.

2 Transducteurs sous-séquentiels

2.1 Définition et exemples

Question 17 Il s'agit juste de rajouter un 1 en sortie si la retenue fait augmenter le nombre total de bit. On pose $\rho(q_0) = 1$ et $\rho(q_1) = \rho(q_2) = \varepsilon$.

Question 18 On propose :



2.2 Langages rationnels

Question 19 Quitte à rajouter des états, on peut remplacer une transition étiquetée par un mot non vide $p \xrightarrow{u=a_1\ldots a_n} q$ par plusieurs transitions étiquetées par des lettres $p = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n = q$. De plus, on peut supprimer les transitions étiquetées par ε (résultat classique au programme).

Question 20 Soit L un langage rationnel, reconnu par un automate A. Avec les définitions précédentes, l'automate B reconnaît bien $\varphi_T(L(A))$.

En effet, on peut prouver par induction que $(\delta(q_0, u), \delta_A(q_A, u)) \in \Delta_B(q_B, \lambda(u))$. De plus, si on ajoute la lecture de $\rho(\delta(q_0, u))$, on en déduit que $(q_f, \delta_A(q_A, u)) \in \Delta_B(q_B, \varphi_T(u))$.

Par définition de F_B, si $u \in L(A)$ alors $\varphi_T(u) \in L(B)$.

Réciproquement, si $v \in L(B)$, il existe un calcul dans B de la forme :

$$(q_0, q_A) = (q_0, p_0) \xrightarrow{\lambda(q_0,a_1)} (q_1, p_1) \xrightarrow{\lambda(q_1,a_2)} \ldots \xrightarrow{\lambda(q_{n-1},a_n)} (q_n, p_n) \xrightarrow{\rho(q_n)} (q_f, p_n)$$

avec $v = \lambda(q_0, a_1)\lambda(q_1, a_2)\ldots\lambda(q_{n-1}, a_n)\rho(q_n)$ et $p_n \in F_A$. Si on pose $u = a_1\ldots a_n$, on a bien, par définition, $v = \varphi_T(u)$ et $u \in L(A)$, car $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_1} \ldots \xrightarrow{a_1} p_n$ est un calcul acceptant dans A.

Question 21 Par l'absurde, supposons qu'il existe $m < n$ avec $\delta_A(q_A, a^m) = \delta_A(q_A, a^n) = q$. Alors :

- sachant que $a^m b^m \in L_0$, $\delta_A(q, b^m) = \delta_A(q_A, a^m b^m) \in F_A$;
- sachant que $a^n b^m \notin L_0$, $\delta_A(q, b^m) = \delta_A(q_A, a^n b^n) \notin F_A$.

L'état $\delta_A(q, b^m)$ doit donc être à la fois final et non final, ce qui est absurde.

Dès lors, la fonction $n \mapsto \delta_A(q_A, a^n)$ est une injection de $\mathbb{N}$ dans Q_A, donc le cardinal de Q_A est infini, ce qui est contradictoire avec le fait que A est un automate fini. On en déduit par l'absurde que L_0 n'est pas rationnel.

Question 22 On considère le transducteur sous-séquentiel à un seul état q_0 défini par $T = (\{q_0\}, \Sigma, \Gamma, \delta, \lambda, q_0, \rho)$ où :

- pour $a \in \Sigma$, $\delta(q_0, a) = q_0$ et $\lambda(q_0, a) = \varepsilon$;
- $\rho(q_0) = \varepsilon$.

Il est clair que pour tout langage L non vide, $\varphi_T(L) = \{\varepsilon\}$ est un langage rationnel, et en particulier $\varphi_T(L_0)$ est rationnel. Pourtant, d'après la question précédente, L_0 n'est pas rationnel.

Question 23 Soit $B = (Q_B, \Gamma, \delta_B, q_B, F_B)$ un automate fini déterministe complet reconnaissant L. Pour simplifier, commençons par montrer le résultat dans le cas où T est séquentiel (c'est-à-dire lorsque $\rho(q) = \varepsilon$ pour tout q).

On pose $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ où :

- $Q_A = Q \times Q_B$;
- $q_A = (q_0, q_B)$;
- $F_A = Q \times F_B$;
- pour $q \in Q$, $p \in Q_B$, $a \in \Sigma$, $\delta_A((q,p),a) = (\delta(q,a), \delta_B(p, \lambda(q,a)))$.

Alors on peut montrer (mais ce n'était pas attendu par l'énoncé) que pour $u \in \Sigma^*$, $\delta_A(q_A, u) = (\delta(q,u), \delta_B(q_B, \lambda(q_0,u)))$. Ainsi :

$$u \in \varphi_T^{-1}(L) \iff \varphi_T(u) \in L \iff \delta_B(q_B, \varphi_T(u)) \in F_B \iff \delta_A(q_A,u) \in F_A \iff u \in L(A)$$

$\varphi_T^{-1}(L)$ est bien rationnel.

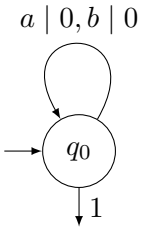
Dans le cas où T est sous-séquentiel, on garde la même définition, en changeant les états finaux en :

$$F_A = \{(q,p) \in Q \times Q_B \mid \exists u \in \Sigma^*, \delta_A(q_A,u) = (q,p) \text{ et } \delta_B(p, \rho(q)) \in F_B\}$$

2.3 Fonctions sous-séquentielles

Question 24

On considère le transducteur T sous-séquentiel suivant :



Alors $\varphi_T(a) = 01$ et $\varphi_T(aa) = 001$. Or, a est préfixe de aa, mais 01 n'est pas préfixe de 001. φ_T ne conserve donc pas les préfixes.

Question 25 On commence par remarquer qu'une fonction séquentielle f est un cas particulier de fonction sous-séquentielle, où on a posé $\rho(q) = \varepsilon$ pour tout état q d'un transducteur séquentiel de fonction f.

Montrons que f conserve les préfixes. Pour cela, soit $T = (Q, \Sigma, \Gamma, \delta, \lambda, q_0)$ un transducteur séquentiel tel que $\varphi_T = f$. Soit $u, v \in \Sigma^*$. Alors $\varphi(u)$ est préfixe de $\varphi(uv)$. En effet, $\varphi(uv) = \lambda(q_0, uv) = \lambda(q_0,u)\lambda(\delta(q_0,u), v) = \varphi(u)\lambda(\delta(q_0,u),v)$.

Question 26 Avec les notations de l'énoncé, soit $u \in \Sigma^*$ tel que $q = \delta(q_0, u)$. Comme u est un préfixe de ua, $\varphi_T(u)$ est un préfixe de $\varphi_T(ua)$, par hypothèse, donc il existe $v \in \Gamma^*$ tel que $\varphi_T(u)v = \varphi_T(ua)$.

Or, $\varphi_T(u) = \lambda(q_0, u)\rho(q)$ et $\varphi_T(ua) = \lambda(q_0, ua)\rho(\delta(q, a)) = \lambda(q_0, u)\lambda(q, a)\rho(\delta(q, a))$. On a alors $\varphi_T(u) = \lambda(q_0, u)\rho(q)v = \lambda(q_0, u)\lambda(q, a)\rho(\delta(q, a))$, soit $\rho(q)v = \lambda(q, a)\rho(\delta(q, a))$.

Question 27 Montrons par induction que pour $q \in Q$ et $u \in \Sigma^*$, on a :

$$\rho(q)\lambda'(q, u) = \lambda(q, u)\rho(\delta(q, u))$$

- pour $u = \varepsilon$, $\lambda'(q, u) = \lambda(q, u) = \varepsilon$, et $\delta(q, u) = q$, d'où l'égalité ;
- supposons le résultat établi pour u, et soit $a \in \Sigma$. Alors :

$$\begin{aligned}
\rho(q)\lambda'(q, ua) &= \rho(q)\lambda'(q, u)\lambda'(\delta(q, u), a) \\
&= \lambda(q, u)\rho(\delta(q, u))\lambda'(\delta(q, u), a) \text{ par hypothèse d'induction} \\
&= \lambda(q, u)\lambda(\delta(q, u), a)\rho(\delta(\delta(q, u), a)) \text{ par la question précédente} \\
&= \lambda(q, ua)\rho(\delta(q, ua)) \text{ par définition de } \lambda \text{ et } \delta
\end{aligned}$$

On conclut par induction.

Question 28 Soit T un transducteur sous-séquentiel tel que φ_T conserve les préfixes. Quitte à supprimer des états, on peut supposer que tous les états sont accessibles. Soit $T' = (Q, \Sigma, \Gamma, \delta, \lambda', q_0)$ un transducteur séquentiel. Alors $\varphi_{T'} = \varphi_T$; en effet, par la question précédente, pour $u \in \Sigma^*$, $\rho(q_0)\lambda'(q_0, u) = \lambda(q_0, u)\rho(\delta(q_0, u)) = \varphi_T(u)$. De plus, $\rho(q_0) = \varepsilon$, car φ_T conserve les préfixes, donc $\varphi_T(\varepsilon) = \rho(q_0) = \varepsilon$. On en déduit que $\varphi_{T'}(u) = \lambda'(q_0, u) = \varphi_T(u)$.

3 Théorèmes de Ginsburg-Rose et Choffrut

3.1 Distance préfixe

Question 29 On montre les différents points :

- comme $u \wedge v = v \wedge u$ et que l'addition est commutative, $d(u, v) = d(v, u)$;
- $u \wedge v$ étant un préfixe de u, $|u| - |u \wedge v| \geqslant 0$. De même, $|v| - |u \wedge v| \geqslant 0$, soit $d(u, v) \geqslant 0$;
- si $d(u, v) = 0$, alors $2|u \wedge v| = |u| + |v|$. Sachant que $|u \wedge v| \leqslant |u|$ et $\leqslant |v|$, on en déduit que $|u \wedge v| = |u| = |v|$. u et v ont donc un plus long préfixe commun qui leur est égal, donc sont égaux ;
- On remarque que $d(u, v) + d(v, w) - d(u, w) = |u \wedge w| + |v| - |u \wedge v| - |v \wedge w|$. Distinguons :
  - si $|u \wedge w| \geqslant |u \wedge v|$ (ou de même si $|u \wedge w| \geqslant |v \wedge w|$), alors sachant que $|v| \geqslant |v \wedge w|$, on en déduit $d(u, v) + d(v, w) - d(u, w) \geqslant 0$;
  - sinon, sans perte de généralité, $|u \wedge w| < |u \wedge v| \leqslant |v \wedge w|$. Alors $u \wedge v$ est un préfixe de u et de $v \wedge w$, donc de w. On en déduit que $|u \wedge v| \leqslant |u \wedge w|$, ce qui est absurde.

Question 30 Par définition de u, pour $v \in X$, il existe $w \in X$ tel que $v \wedge w = u$ (sinon il y aurait un préfixe commun plus long que u). On a alors $d(u, v) = d(v \wedge w, v) \leqslant d(v, w) \leqslant \max_{x,y \in X} d(x, y)$.

Question 31 Soit f une fonction séquentielle. On a déjà montré les points (a) et (b) dans des questions précédentes. Si on suppose vrai le théorème de Choffrut, alors le point (c) est également vérifié.

Réciproquement, si f vérifie (b) et (c), alors f est sous-séquentielle. Si elle vérifie de plus (a), alors elle est séquentielle, d'après la question 28.

Question 32 Avec les notations de l'énoncé, soient $u, v \in \Sigma^*$. Notons $w = u \wedge v$ et u', v' tels que $u = wu'$, $v = wv'$, et $q = \delta(q_0, w)$. Alors $f(u) = \lambda(q_0, u)\rho(\delta(q_0, u)) = \lambda(q_0, w)\lambda(q, u')\rho(\delta(q_0, u))$. On a donc $|f(u)| = |\lambda(q_0, w)| + |u'|K_1 + K_2$ (et de même pour v).

On a alors :

$$\begin{array}{rcl} d(f(u), f(v)) & = & |f(u)| + |f(v)| - 2|f(u) \wedge f(v)| \\ & \leqslant & |f(u)| + |f(v)| - 2|\lambda(q_0, u)| \text{ car } \lambda(q_0, u) \text{ est un préfixe de } f(u) \wedge f(v) \\ & = & K_1(|u'| + |v'|) + 2K_2 \\ & = & K_1 d(u, v) + 2K_2 \\ & \leqslant & K d(u, v) \end{array}$$

Cette dernière inégalité est bien vérifiée, car elle est vrai si $d(u, v) = 0$ (car alors toutes les valeurs valent 0) et reste vraie si $d(u, v) \geqslant 1$ (car alors $2K_2 \leqslant 2K_2 d(u, v)$).

$\star\star\star$