Ce devoir est composé de 5 exercices indépendants.

I Exercice simple de déduction naturelle

On rappelle les règles de déduction naturelle classique :

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to_i \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B} \to_e \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_i \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_e^g \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_e^d \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \bot_e$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_i^g \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_i^d \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C \quad \Gamma \vdash A \vee B}{\Gamma \vdash C} \vee_e \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg_i \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \bot} \neg_e$$

En les utilisant, montrer les séquents suivants :

1. $A \wedge B \vdash A \longrightarrow B$

   Solution :

   $$\dfrac{\dfrac{\dfrac{}{A \wedge B, A \vdash A \wedge B}\,\text{ax}}{A \wedge B, A \vdash B}\,\wedge_e}{A \wedge B \vdash A \longrightarrow B}\,\longrightarrow_i$$

2. $A \vee B \vdash B \vee A$

   Solution :

   $$\dfrac{\dfrac{}{A \vee B \vdash A \vee B}\,\text{ax} \qquad \dfrac{\dfrac{}{A \vee B, A \vdash A}\,\text{ax}}{A \vee B, A \vdash B \vee A}\,\vee_i \qquad \dfrac{\dfrac{}{A \vee B, B \vdash B}\,\text{ax}}{A \vee B, B \vdash B \vee A}\,\vee_i}{A \vee B \vdash B \vee A}\,\vee_e$$

3. $\vdash \neg(A \wedge \neg A)$

   Solution :

   $$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge \neg A \vdash A \wedge \neg A}\,\text{ax}}{A \wedge \neg A \vdash A}\,\wedge_e \qquad \dfrac{\dfrac{}{A \wedge \neg A \vdash A \wedge \neg A}\,\text{ax}}{A \wedge \neg A \vdash \neg A}\,\wedge_e}{A \wedge \neg A \vdash \bot}\,\neg_e}{\vdash \neg(A \wedge \neg A)}\,\neg_i$$

II Connecteur de Sheffer (d'après CCP 2020)

Remarque : On confond dans cet exercice équivalence et égalité de formule.

Soit $x_1, \cdots, x_n$ un ensemble de n variables propositionnelles. On appelle minterme toute formule de la forme $y_1 \wedge y_2 \wedge \cdots y_n$ où pour tout $i \in \{1, \cdots, n\}$, y_i est un élément de $\{x_i, \neg x_i\}$. On appelle maxterme toute formule de la forme $y_1 \vee y_2 \vee \cdots y_n$ où pour tout $i \in \{1, \cdots, n\}$, y_i est un élément de $\{x_i, \neg x_i\}$.

Les mintermes (respectivement maxtermes) $y_1 \wedge y_2 \wedge \cdots y_n$ et $y'_1 \wedge y'_2 \wedge \cdots y'_n$ (resp $y_1 \vee y_2 \vee \cdots y_n$ et $y'_1 \vee y'_2 \vee \cdots y'_n$) sont considérés identiques si les ensembles $\{y_i, 1 \leq i \leq n\}$ et $\{y'_i, 1 \leq i \leq n\}$ le sont.

1. Donner l'ensemble des mintermes et des maxtermes sur l'ensemble $\{x_1, x_2\}$.

   Solution : Les mintermes sont $\{x_1 \wedge x_2, x_1 \wedge \neg x_2, \neg x_1 \wedge x_2, \neg x_1 \wedge \neg x_2\}$ et les maxtermes sont $\{x_1 \vee x_2, x_1 \vee \neg x_2, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2\}$.

Soit ϕ une formule propositionnelle qui s'écrit à l'aide de n variables propositionnelles $x_1, \cdots, x_n$.
On appelle forme normale conjonctive de ϕ toute conjonction de maxtermes logiquement équivalente à ϕ.
On appelle forme normale disjonctive de ϕ toute disjonction de mintermes logiquement équivalente à ϕ.

Un ensemble de connecteurs logiques C est un système complet si toute formule propositionnelle est équivalente à une formule n'utilisant que les connecteurs de C.

Par définition, $C = \{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$ est un système complet.

On définit le connecteur de Sheffer, ou d'incompatibilité, par $x_1 \diamond x_2 = \neg x_1 \vee \neg x_2$.

2. Construire la table de vérité du connecteur de Sheffer.

> **Solution** :
>
> | x_1 | x_2 | $x_1 \diamond x_2$ |
> |---|---|---|
> | 0 | 0 | 1 |
> | 0 | 1 | 1 |
> | 1 | 0 | 1 |
> | 1 | 1 | 0 |

3. Exprimer ce connecteur en fonction de $\neg$ et $\wedge$.

> **Solution** : vue la table de vérité, $x_1 \diamond x_2 = \neg(x_1 \wedge x_2)$.

4. Vérifier que $\neg x_1 = x_1 \diamond x_1$.

> **Solution** :
>
> - Si $x_1 = 0$, alors $\neg x_1 = 1$ et $x_1 \diamond x_1 = 1$.
> - Si $x_1 = 1$, alors $\neg x_1 = 0$ et $x_1 \diamond x_1 = 0$.
>
> Donc $\neg x_1 = x_1 \diamond x_1$.

5. En déduire une expression des connecteurs $\wedge, \vee$ et $\Rightarrow$ en fonction du connecteur de Sheffer. Justifier en utilisant des équivalences avec les formules propositionnelles classiques.

> **Solution** :
>
> - $\wedge$: $x_1 \wedge x_2 = \neg(\neg x_1 \vee \neg x_2) = \neg(x_1 \diamond x_2) = (x_1 \diamond x_2) \diamond (x_1 \diamond x_2)$.
> - $\vee$: $x_1 \vee x_2 = (\neg x_1) \diamond (\neg x_2) = (x_1 \diamond x_1) \diamond (x_2 \diamond x_2)$.
> - $\Rightarrow$: $x_1 \Rightarrow x_2 = (\neg x_1) \vee x_2 = x_1 \diamond \neg x_2 = x_1 \diamond (x_2 \diamond x_2)$.

6. Donner une forme normale conjonctive de la formule $x_1 \diamond x_2$.

> **Solution** : $x_1 \diamond x_2 = \neg x_1 \vee \neg x_2$.

7. Donner de même une forme normale disjonctive de la formule $x_1 \diamond x_2$.

> **Solution** : $(\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge \neg x_2)$.

8. Démontrer par induction sur les formules propositionnelles que l'ensemble de connecteurs $C = \{\diamond\}$ est un système complet.

> **Solution** : Montrons par induction que si ϕ est une formule propositionnelle, alors ϕ est équivalente à une formule n'utilisant que le connecteur de Sheffer.
>
> - **Cas de base** : Si ϕ est une variable propositionnelle, alors ϕ n'utilise pas de connecteur et est équivalente à elle-même.
> - **Hérédité** : Soit ϕ une formule qui n'est pas une variable propositionnelle. Si $\phi = \neg\phi_1$ alors, par induction, il existe une formule ψ_1 équivalente à ϕ_1 et n'utilisant que le connecteur de Sheffer. Alors $\phi = \psi_1 \diamond \psi_1$ s'écrit en utilisant utilisant que le connecteur de Sheffer.
> Sinon, ϕ est de la forme $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$ ou $\neg\phi_1$. Par hypothèse d'induction, il existe des formules ψ_1 et ψ_2 n'utilisant que le connecteur de Sheffer telles que ϕ_1 et ϕ_2 soient équivalentes à ψ_1 et ψ_2. D'après la question 5, ϕ s'écrit aussi en utilisant que le connecteur de Sheffer.
>
> Donc $C = \{\diamond\}$ est un système complet.

9. Application : soit $\phi = x_1 \vee (\neg x_2 \wedge x_3)$. Donner une forme logiquement équivalente de ϕ utilisant uniquement le connecteur de Sheffer.

> **Solution** :
>
> $$\begin{aligned}\phi &= x_1 \vee (\neg x_2 \wedge x_3)\\ \phi &= x_1 \vee ((x_2 \diamond x_2) \wedge x_3)\\ \phi &= x_1 \vee x\\ \phi &= (x_1 \diamond x_1) \diamond (x \diamond x)\end{aligned}$$
>
> où $x = (((x_2 \diamond x_2) \diamond x_3) \diamond ((x_2 \diamond x_2) \diamond x_3))$.

III Mots de Dyck (d'après Epita 2022)

On appelle mot de Dyck un mot m sur l'alphabet $\{a, b\}$ tel que m contient autant de a que de b et tout préfixe m contient au moins autant de a que de b.

Par exemple, $m = aababb$ est un mot de Dyck, car il possède trois a et trois b, et ses préfixes sont ε (le mot vide), a, aa, aab, $aaba$, $aabab$ et m, et aucun ne contient strictement plus de b que de a.

1. Parmi les mots suivants, indiquer ceux qui sont de Dyck. On ne demande pas de preuve.

$$\varepsilon \quad aabbbaab \quad aaab \quad abab \quad aaabbb$$

> Solution : ε, $abab$ et $aaabbb$.
> Remarque : si on remplace a par (et b par), les mots de Dyck sont les mots bien parenthésés.

On représente dans la suite en OCaml un mot sur l'alphabet $\{a, b\}$ par la liste de ses lettres. On définit les types suivants :

```
type lettre = A | B

type mot = lettre list
```

Le mot $m = aababb$ sera donc représenté par la liste `[A; A; B; A; B; B]`.

2. Écrire une fonction `verifie_dyck` de type `mot -> bool` renvoyant un booléen indiquant si le mot passé en entrée est de Dyck.

> Solution : On peut utiliser une variable (en argument) pour se souvenir du nombre de `A` déjà vu moins le nombre de `B`.
>
> ```
> let verifie_dyck m =
>     let rec aux n m = match m with
>         | [] -> n = 0
>         | A::q -> aux (n + 1) q
>         | B::q -> n > 0 && aux (n - 1) q in
>     aux 0 m
> ```

On admet la propriété suivante : tout mot m de Dyck non vide se décompose de manière unique sous la forme $m = aubv$, où u et v sont des mots de Dyck. On pourra utiliser sans démonstration la caractérisation suivante : aub est le plus petit préfixe non vide de m contenant autant de a que de b.

3. Donner sans démonstration la décomposition de chacun des mots de Dyck suivants :

$$ab \quad aababb \quad ababab \quad aabbab$$

> Solution :
>
> - $ab = aubv$ avec $u = v = \varepsilon$.
> - $aababb = aubv$ avec $u = abab$ et $v = \varepsilon$.
> - $ababab = aubv$ avec $u = \varepsilon$ et $v = abab$.

- $aabbab = aubv$ avec $u = ab$ et $v = ab$.

4. Écrire une fonction `decompo_dyck` de type `mot -> mot * mot`, prenant en entrée un mot m supposé non vide et de Dyck (il est inutile de le vérifier), et renvoyant le couple de mots de Dyck (u, v) tel que $m = aubv$.

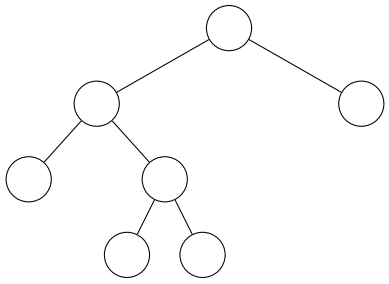
   Solution : On fait le même comptage qu'en Q2, en s'arrêtant lorsqu'on trouve un préfixe avec le même nombre de `A` et de `B` (`if n = 1`). `List.tl` sert à enlever le `A` initial.

```
let decompo_dyck m =
  let rec aux n m = match m with (* renvoie (u, v) tels que m = aubv avec u et v de Dyck*)
        | [] -> failwith "decompo_dyck"
        | A::q ->
          let uq, vq = aux (n + 1) q in
          A::uq, vq
        | B::q ->
          if n = 1 then [], q
          else
            let uq, vq = aux (n - 1) q in
            B::uq, vq in
  let u, v = aux 0 m in
  List.tl u, v
```

Il existe une bijection naturelle entre les arbres binaires stricts (tout nœud de l'arbre possède 0 ou 2 fils) et les mots de Dyck, basée sur la décomposition précédente :

- au mot vide est associé l'arbre réduit à une feuille;
- à un mot de Dyck non vide $aubv$ où u et v sont de Dyck, on associe l'arbre binaire où le sous-arbre gauche est associé au mot u et le sous-arbre droit au mot v.

5. Dessiner l'arbre associé au mot de Dyck $m = aababb$. Les nœuds ne portent pas d'étiquettes. Solution :



On définit le type suivant :

```
type arbre = F | N of arbre * arbre;;
```

6. Écrire une fonction `mot_a_arbre` de type `mot -> arbre` renvoyant l'arbre binaire strict associé à un mot de Dyck (on ne vérifiera pas que le mot passé en entrée est bien de Dyck).

   Solution :

```
let rec mot_a_arbre m = match m with
  | [] -> F
  | _ ->
    let u, v = decompo_dyck m in
    N(mot_a_arbre u, mot_a_arbre v)
```

7. Écrire une fonction `arbre_a_mot` de type `arbre -> mot` faisant l'inverse.

   Solution :

```
let rec arbre_a_mot a = match a with
  | F -> []
  | N(u, v) -> A::(arbre_a_mot u)@(B::(arbre_a_mot v))
```

8. Montrer que le language L des mots de Dyck n'est pas rationnel.

   Solution : Supposons que L soit rationnel. Soit $n \in \mathbb{N}$ donné par le lemme de l'étoile.
   Soit $m = a^n b^n$. Comme $m \in L$ et $|m| \geq n$, il existe x, y, z tels que $|xy| \leq n$, $y \neq \varepsilon$, $m = xyz$ et $xy^*z \subseteq L$.
   Comme $|xy| \leq n$, y ne contient que de a. Comme $y \neq \varepsilon$, y contient au moins un a. Donc xy^2z contient strictement plus de a que de b, ce qui contredit le fait que $xy^2z \in L$: absurde.
   Donc L n'est pas rationnel.

9. Soit $C(n)$ le nombre de mots de Dyck de longueur $2n$. Trouver une équation de récurrence sur $C(n)$, puis montrer que $C(n) = \frac{1}{n+1}\binom{2n}{n}$.

   Solution : Un mot de Dyck de longueur $2n$ se décompose sous la forme $aubv$ où u est un mot de Dyck de longueur $2k$ et v un mot de Dyck de longueur $2(n-k-1)$ pour $k \in [\![0, n-1]\!]$. Il y a donc $C(k)C(n-k-1)$ mots de Dyck de longueur $2n$ de cette forme. On en déduit que $C(n) = \sum_{k=0}^{n-1} C(k)C(n-k-1)$.

   On a $C(0) = 1$ et $C(1) = 1$. On montre alors par récurrence que $C(n) = \frac{1}{n+1}\binom{2n}{n}$.

IV Typage OCaml (d'après CCP MPI)

On souhaite formaliser le typage OCaml. Pour cela, on notera $\Gamma \vdash e : \tau$ si l'expression OCaml e est typée par le type τ et on utilisera les règles suivantes :

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}\ (1) \qquad \frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}}\ (2) \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash n : \texttt{int}}\ (3)$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ (4) \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \texttt{fun}\ x \to e : \sigma\ \to\ \tau}\ (5) \qquad \frac{\Gamma \vdash f : \sigma\ \to\ \tau \quad \Gamma \vdash e : \sigma}{\Gamma \vdash f\ e : \tau}\ (6)$$

1. Soit $\Gamma = \{$`f : a -> (b -> a), g : b -> a`$\}$. Montrer $\Gamma \vdash$ `fun x -> f (g x) x` $: \tau$ pour un certain type τ à déterminer.

   Solution :

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : b \vdash g\ x : a}\ (4) \quad \dfrac{}{\Gamma, x : b \vdash x : b}\ (4)}{\Gamma, x : b \vdash f\ (g\ x)\ x : a}\ (6)}{\Gamma \vdash \texttt{fun}\ x \to f\ (g\ x)\ x : b \to a}\ (5)$$

2. Quelles analogies peut-on faire entre le typage OCaml et la déduction naturelle ?

   Solution :
   (5) est analogue à l'introduction de l'implication en déduction naturelle.

   (6) est analogue à l'élimination de l'implication en déduction naturelle.

3. Montrer que `(fun g -> g 1 2) (fun x -> 3)` n'est pas typable, c'est-à-dire qu'il n'existe pas de type τ tel que $\vdash$ `(fun g -> g 1 2) (fun x -> 3)` $: \tau$ soit prouvable.

   Solution : Supposons qu'il existe un tel type τ.
   Comme x est une application de fonction, la seule règle permettant de prouver $\vdash$ `(fun g -> g 1 2) (fun x -> 3)` : τ est (6) avec $f =$ `fun g -> g 1 2` et $e =$ `fun x -> 3`, ce qui donne $\vdash$ `fun g -> g 1 2` $: \sigma \to \tau$ et $\vdash$ `fun x -> 3` : σ pour un certain σ.
   La seule règle applicable pour typer e est (5) et (3), ce qui donne $\sigma = \sigma' \to$ `int` pour un certain σ'.
   f doit alors être de type $(\sigma' \to$ `int`$) \to \tau$.

A1 – Profondeur de diagrammes de décision

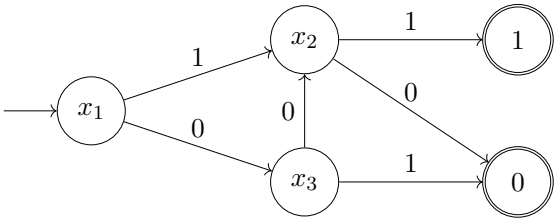
On fixe un ensemble $X = \{x_1, \dots, x_n\}$ de variables booléennes. Un *diagramme de décision* D sur X est la donnée d'un graphe orienté (V, E), supposé sans cycle, d'un nœud initial $v_{\text{init}} \in V$, d'une partition de V en $V = V_0 \sqcup V_1 \sqcup V'$, d'une partition de E en en $E = E_0 \sqcup E_1$, et d'une fonction $\mu : V' \to X$, de sorte que :

— Aucun nœud $v \in V_0$ ou $v \in V_1$ n'a d'arête sortante, i.e., il n'existe aucun $w \in V$ tel que $(v, w) \in E$;

— Tout nœud $v \in V'$ a exactement une arête sortante dans E_0 et une arête sortante dans E_1, i.e., il existe exactement un $w_0 \in V$ et exactement un $w_1 \in V$ tels que $(v, w_0) \in E_0$ et $(v, w_1) \in E_1$.

Si on se donne une *valuation* $\nu : X \to \{0, 1\}$, le diagramme de décision D associe ν à une valeur $b \in \{0, 1\}$ obtenue comme suit : on initialise le nœud courant par $v := v_{\text{init}}$, tant que le nœud courant v est dans V' alors on remplace v par $v := w_0$ ou $v := w_1$ comme défini ci-dessus selon la valeur de $\nu(\mu(v))$, et une fois que $v \in V_0$ ou $v \in V_1$ alors on renvoie 0 ou 1 suivant le cas.

Question 0. On considère $X = \{x_1, x_2, x_3, x_4\}$ et le diagramme D_0 suivant, où on indique dans chaque nœud la valeur de μ ou l'appartenance à V_0 ou V_1, et on indique le nœud initial par une flèche :



À quelle valeur est associée la valuation qui envoie x_1, x_2, x_3, x_4 respectivement vers $1, 1, 0, 1$? vers $0, 1, 0, 0$?

Question 1. Donner une formule logique décrivant la fonction booléenne représentée par D_0.

Question 2. Si on se donne une formule logique ϕ, on dit que D *représente* ϕ si ϕ est la fonction booléenne qu'il décrit. Donner un diagramme de décision D_2 représentant la fonction $\neg(x_1 \wedge x_2) \vee x_3$

Question 3. La *profondeur* d'un diagramme de décision est la plus grande longueur possible d'un chemin orienté à partir du nœud initial, en comptant le nombre de nœuds de V' traversés (y compris v_{init}). Décrire la profondeur du diagramme D_0 et celle du diagramme D_2.

Question 4. Peut-on avoir deux diagrammes de décision de profondeurs différentes qui représentent une même formule logique ?

Question 5. On appelle *profondeur minimale* d'une fonction booléenne ϕ la plus petite profondeur possible pour un diagramme de décision représentant ϕ.

Quelle est la profondeur minimale de la fonction identifiée en question 1 ?

Question 6. Une fonction booléenne ϕ sur n variables est dite *évasive* si sa profondeur minimale est de n. Donner un exemple d'une famille infinie de fonctions évasives, et justifier.

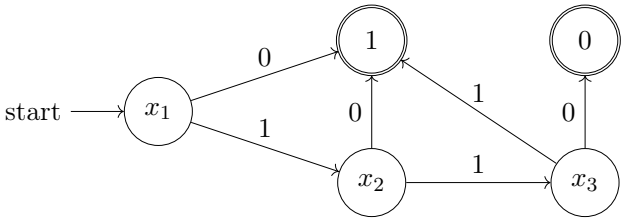
Question 7. On considère, pour tout $n \geq 1$, la fonction booléenne ψ_n définie par la formule $(x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee \cdots \vee (x_{n-1} \wedge x_n)$. Ces fonctions sont-elles évasives ? Justifier.

Corrigé

Question 0. La première valuation est associée à 1 (test x_1 puis x_2), la deuxième est associée également à 1 (test x_1 puis x_3 puis x_2).

Question 1. On peut facilement obtenir une formule en forme normale disjonctive en considérant les chemins vers 1 : $(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_3 \wedge x_2)$. Ou encore, en factorisant puis en simplifiant : $x_2 \wedge (x_1 \vee \neg x_3)$.

Question 2. On peut récrire cela en utilisant la loi de de Morgan : $\neg x_1 \vee \neg x_2 \vee x_3$. Ainsi, par exemple :



Question 3. La profondeur du diagramme D_0 est de 3, celle de D_2 est également de 3 (ce sera toujours au moins 3 quelle que soit la réponse à la question 2, possiblement davantage si le diagramme teste la même variable plusieurs fois sur un même chemin).

Question 4. Manifestement oui : on peut toujours augmenter la profondeur en remplaçant le nœud initial par un test inutile dont les deux branches amènent au même nœud.

Question 5. Clairement cette profondeur est au plus de 3, vu que D_0 témoigne qu'une profondeur 3 est réalisable. Justifions qu'une profondeur de 2 est impossible. Supposons par l'absurde qu'on ait un diagramme D_5 de profondeur 2 représentant cette fonction. Distinguons suivant la variable étiquetant v_{init} :

- Si c'est x_2, alors considérons le sommet obtenu après 1. Il faut à présent vérifier que $x_1 \vee \neg x_3$ est vrai, or on n'a testé aucune de ces deux variables et on a un unique test avant de devoir atteindre une feuille, c'est clairement impossible.
- Si c'est x_1, considérons le sommet obtenu après 0. Il faut à présent vérifier que $x_2 \wedge \neg x_3$ est vrai avec une profondeur restante de 1, et comme précédemment c'est impossible.
- Si c'est x_3, considérons le sommet obtenu après 0. Il faut à présent vérifier que $x_2 \wedge x_1$ est vrai, c'est impossible pour la même raison.
- Le cas x_4 est manifestement inutile vu que la formule ne dépend pas de la valeur de x_4.

Ainsi, par l'absurde, il n'y a pas de diagramme de profondeur 2 représentant la même fonction que D_0, donc la profondeur minimale de cette fonction est bien de 3.

Question 6. *[Indication 1 : qu'est-ce qui fait qu'une fonction booléenne est évasive ?]*

[Indication 2 : en retirant x_4, est-ce que la fonction de la question 0 était évasive ? pourquoi, intuitivement ?]

[Indication 3 : donner la famille : $\phi_n : x_1 \wedge \cdots \wedge x_n$, et demander la preuve. (Cela fonctionne aussi, par dualité, pour $x_1 \vee \cdots \vee x_n$; et également pour $x_1 \otimes \cdots \otimes x_n$.)]

Démonstration du caractère évasif de $\phi_n : x_1 \wedge \cdots \wedge x_n$. Supposons par l'absurde qu'un diagramme de décision de profondeur $< n$ teste ϕ_n. Considérons la valuation où toutes les variables sont vraies, et son chemin du nœud initial vers une feuille, nécessairement étiquetée par 1. Par le principe des tiroirs, il y a une variable, mettons x_i, qui n'est pas testée sur ce chemin. Ainsi, en changeant la valeur de x_i à 0, on a une valuation qui est également envoyée vers 1 par le diagramme, or ϕ_n s'évalue alors à 0, contradiction.

Question 7. La fonction $\psi_1 : x_0 \wedge x_1$ est évasive pour la même raison qu'à la question précédente.

La fonction $\psi_2 : (x_0 \wedge x_1) \vee (x_1 \wedge x_2)$ est également évasive :

— si l'on teste d'abord x_0 et que la réponse est 0, on doit tester $x_1 \wedge x_2$ qui est évasive

— même raisonnement pour x_2

— si l'on teste d'abord x_1 et que la réponse est 1, on doit tester $x_0 \vee x_2$, qui est évasive.

En revanche, de façon surprenante, la fonction $\psi_3 : (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$ n'est *pas* évasive ! En effet :

— On teste d'abord x_1. Si la réponse est 0, alors on doit tester $x_2 \wedge x_3$ et on n'a pas besoin de tester x_0.

— Si on a $x_1 = 1$, alors il reste à tester $x_0 \vee x_2 \vee (x_2 \wedge x_3)$, ce qui se simplifie en $x_0 \vee x_2$, et on n'a pas besoin de tester x_3.

On peut facilement montrer par récurrence que, pour n multiple de 3, la fonction ψ_n n'est pas évasive. En effet, le cas de base est traité ci-dessus. Ensuite :

— On teste d'abord x_1. Si la réponse est 0, alors on doit tester le reste de la fonction mais sans dépendre de x_0.

— Si on a $x_1 = 1$, alors il reste à tester $x_0 \vee x_2 \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4) \vee \cdots$, ce qui se simplifie en $x_0 \vee x_2 \vee (x_3 \wedge x_4) \vee \cdots$, et quitte à tester x_0 et x_2, et à renommer les variables, on est ramené au cas ψ_{n-3} qui n'est pas évasive par récurrence.

On peut ensuite montrer par récurrence avec prédécesseurs que, pour tout autre n, la fonction ψ_n est évasive. En effet, pour tout x_i que l'on teste en premier :

— Si la réponse est 0, alors on doit tester $(x_0 \wedge x_1) \vee \cdots \vee (x_{i-2} \wedge x_{i-1}) \vee (x_{i+1} \wedge x_{i+2}) \vee \cdots \vee (x_{n-1} \wedge x_n)$. On est donc ramené au cas de ψ_{i-1} et ψ_{n-i-1}.

— Si la réponse est 1, alors on doit tester $(x_0 \wedge x_1) \vee \cdots \vee (x_{i-2} \wedge x_{i-1}) \vee x_{i-1} \vee x_{i+1} \vee (x_{i+1} \wedge x_{i+2}) \vee \cdots \vee (x_{n-1} \wedge x_n)$ ce qui se simplifie en $(x_0 \wedge x_1) \vee \cdots \vee (x_{i-3} \wedge x_{i-2}) \vee x_{i-1} \vee x_{i+1} \vee (x_{i+2} \wedge x_{i+3}) \vee \cdots \vee (x_{n-1} \wedge x_n)$ et on est donc ramené au cas de x_{i-1}, x_{i+1}, ψ_{i-2} et ψ_{n-i-2}.

Ainsi, si ni $i-1$ ni $n-i-1$ ne sont multiples de 3, on considère le cas d'une réponse 0, et on se ramène à $\psi_{i-1} \vee \psi_{n-i-1}$, qui sont sur des variables différentes : toutes deux sont évasives par hypothèse de récurrence avec prédécesseurs, et on montre aisément que la disjonction de deux fonctions évasives sur des variables disjointes est elle-même évasive.

Si l'un de $i-1$ ou $n-i-1$ est multiple de 3, alors on considère le cas d'une réponse 1, et on se ramène à $x_{i-1} \vee x_{i+1} \vee \psi_{i-2} \vee \psi_{n-i-2}$. On fait alors une disjonction de cas :

— Si n est congru à 1 modulo 3, alors :

  — Si $i-1$ est multiple de 3 alors $i-2$ est congru à 2 modulo 3, et $n-i-2$ est congru à $1-1-2 = 1 \bmod 3$.

  — Si $n-i-1$ est multiple de 3 alors $n-i-2$ est congru à 2 modulo 3, et $i-2$ est congru à $-(n-i-2)+n-4 \bmod 3$ c'est-à-dire $1+1-4 = 1 \bmod 3$.

— Si n est congru à 2 modulo 3, alors :

  — Si $i-1$ est multiple de 3 alors $i-2$ est congru à 2 modulo 3, et $n-i-2$ est congru à $2-1-2 = 2 \bmod 3$.

  — Si $n-i-1$ est multiple de 3 alors $n-i-2$ est congru à 2 modulo 3, et $i-2$ est congru à $1+2-4 = 2 \bmod 3$.

— Comme n n'est pas multiple de 3 par hypothèse, cette disjonction de cas est exhaustive.

A8 – Automates à pile et lemme de pompage

Soit Σ un alphabet fini, et Γ un autre alphabet fini appelé *alphabet de pile*. Un *automate à pile* sur Σ est un quintuplet $A = (Q, q_0, \gamma_0, \delta, F)$, où Q est un ensemble fini d'*états*, $q_0 \in Q$ est l'*état initial*, $\gamma_0 \in \Gamma$ est le *symbole de pile initial*, δ est une *fonction de transition* de $Q \times \Sigma \times \Gamma$ vers l'ensemble des parties de $Q \times \Gamma^*$, et $F \subseteq Q$ est un ensemble d'états *finaux*.

Une *configuration* de A est un couple (q, z) où $q \in Q$ est l'*état* et où z est un mot non-vide sur Γ appelé *pile*. La *configuration initiale* est (q_0, γ_0), et une configuration (q, z) est *acceptante* si $q \in F$. Lorsque l'automate est dans une configuration (q, z) et lit une lettre $a \in \Sigma$, il décompose $z = z'\gamma$ avec $\gamma \in \Gamma$ le *sommet de pile*, il choisit un $(q', g) \in \delta(q, a_{i+1}, \gamma)$ tel que $z'g$ soit non-vide, et il aboutit à la configuration $(q', z'g)$. L'automate à pile A *accepte* un mot de Σ^* s'il peut lire ses lettres dans l'ordre à partir de la configuration initiale pour parvenir à une configuration acceptante suivant ces règles.

Question 0. Étant donné un automate A sur Σ sans pile, expliquer comment construire un automate à pile A' sur Σ qui reconnaît le même langage que A.

Question 1. On prend dans cette question $\Sigma = \{a, b\}$. Proposer un automate à pile qui reconnaît le langage $\{a^n b^n \mid n \geq 2\}$. Qu'en déduire ?

Question 2. On prend toujours $\Sigma = \{a, b\}$. Proposer un automate à pile qui reconnaît le langage $\{w \in \Sigma^* \mid |w|_a = |w|_b\}$, où $|w|_a$ et $|w|_b$ désignent respectivement le nombre de a et de b de w.

Suite des questions

Question 3. Pour $\eta \in \mathbb{N}^*$, étant donné un mot w de longueur n et un automate à pile A, les *configurations η-tronquées* sont les triplets (q, z) où $q \in Q$ et z est un mot non-vide sur Γ de longueur $\leq \eta$. Un *calcul η-tronqué* de A sur un mot est une séquence de configurations η-tronquées : la définition est comme auparavant sauf que, si $|z'g| \geq \eta$, on le remplace par son suffixe de longueur η. Le *langage η-tronqué* $L_{\leq\eta}(A)$ de A est le langage des mots sur lesquels A a un calcul η-tronqué acceptant.

Quelle est la relation entre $L_{\leq\eta}(A)$ et $L(A)$? Montrer que, pour tout automate à pile A et $\eta \in \mathbb{N}^*$, le langage η-tronqué de A est un langage régulier, et expliquer comment construire un automate sans pile qui le reconnaît.

Question 4. Soit w un mot de longueur n et A un automate à pile. Soit $\chi = (q_0, z_0), \ldots, (q_p, z_p)$ un calcul de A sur w, et notons $h_i := |z_i|$ pour tout $0 \leq i \leq p$. Pour $\eta \in \mathbb{N}^*$, une *η-montagne* de χ est un triplet d'indices $0 \leq l < m < r \leq p$ tels qu'on ait $h_l = h_r$, on ait $h_m - h_l \geq \eta$, et on ait $h_j > h_l$ pour tout $l < j < r$. Une *montagne* est une η-montagne pour un certain $\eta \in \mathbb{N}^*$.

On appelle G la plus grande longueur d'un mot de Γ^* dans une image de la fonction de transition δ. Montrer que, pour tout $\eta > G$, si $w \in L(A) \setminus L_{\leq\eta}(A)$, alors tout calcul acceptant de A sur w a une $(\eta - G)$-montagne.

Question 5. L'*état de base* d'une η-montagne $l < r$ dans un calcul est le triplet $\beta(l, r) := (q_l, q_r, \gamma)$ où q_l, q_r sont les états des étapes l et r du calcul respectivement, et γ est le dernier symbole de la pile z_l.

Montrer que, pour tout automate à pile A, il existe $\eta_0 \in \mathbb{N}^*$ avec la propriété suivante : pour tout mot w, pour tout calcul χ de A sur w, si χ a une η_0-montagne (l, m, r), alors il existe une montagne (l', m, r') et une montagne (l'', m, r'') avec $l < l' < l'' < m < r'' < r' < r$ telles que $\beta(l', r') = \beta(l'', r'')$.

Question 6. Déduire des trois questions précédentes une forme affaiblie du *lemme de pompage pour les automates à pile* : pour tout langage L reconnu par un automate à pile, il existe un entier $p \in \mathbb{N}$ tel que, pour tout mot $w \in L$ avec $|w| > p$, on peut écrire $w = uvxyz$ où les mots u, v, x, y, z satisfont :

— $|vy| \geq 1$

— Pour tout $n \in \mathbb{N}$, on a $uv^n xy^n z \in L$.

Corrigé

Question 0. On choisit un alphabet de pile Γ arbitraire, et $\gamma_0 \in \Gamma$ arbitraire. Soit $A = (Q, q_0, F, \delta)$ un automate sans pile sur Σ, non nécessairement complet ou déterministe, où $\delta : Q \times \Sigma \to 2^Q$. On définit l'automate $A' = (Q, q_0, \gamma_0, \delta', F)$, où on définit $\delta(q, a, \gamma) := \{(q', \gamma_0) \mid q' \in \delta(q)\}$ pour tous $q \in Q$, $a \in \Sigma$, et $\gamma \in \Gamma$.

Observons d'abord que, dans toutes les configurations accessibles pour l'automate A', la *pile* est exactement γ_0. En effet, à chaque transition, on la laisse à γ_0. De plus, la valeur du sommet de pile n'a jamais d'impact sur les transitions. Ainsi, les séquences de configurations possibles de A' sur n'importe quel mot $w \in \Sigma^*$ correspondent exactement aux séquences d'états possibles de A sur w en ajoutant une pile qui vaut toujours γ_0. En particulier, la configuration initiale correspond à l'état initial, les configurations acceptantes correspondent aux états finaux, et les transitions entre configurations correspondent aux transitions de A. Ainsi, A et A' reconnaissent le même langage.

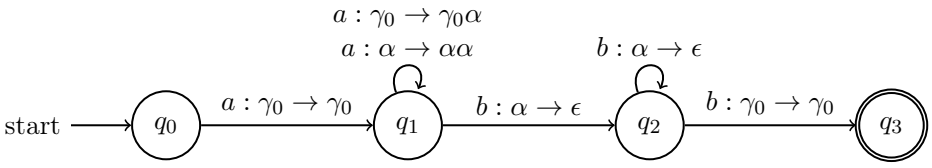
Question 1. On choisit l'alphabet de pile $\Gamma = \{\alpha, \gamma_0\}$. On définit l'automate $A = (\{q_0, q_1, q_2, q_3\}, q_0, \gamma_0, \delta, \{q_3\})$ avec δ comme suit (c'est $\emptyset$ pour les cas non spécifiés) :

— $\delta(q_0, a, \gamma_0) := \{(q_1, \gamma_0)\}$

— $\delta(q_1, a, \gamma) := \{(q_1, \gamma\alpha)\}$ pour $\gamma \in \{\gamma_0, \alpha\}$;

— $\delta(q_1, b, \alpha) := \{(q_2, \epsilon)\}$

— $\delta(q_2, b, \alpha) := \{(q_2, \epsilon)\}$;
— $\delta(q_2, b, \gamma_0) := \{(q_3, \gamma_0)\}$.
Graphiquement :



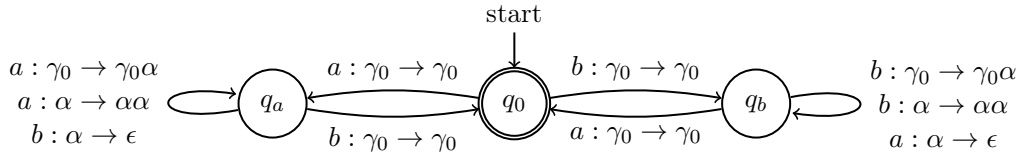
Intuitivement, l'automate passe dans l'état 1 quand il voit un a ; il reste dans l'état 1 tant qu'il lit des a et empile un α par a lu. Lorsqu'il lit des b, il passe dans l'état 1 et dépile un α par b lu. Il atteint l'état final lorsque le fond de la pile γ_0 est atteint en lisant le dernier b, auquel cas il accepte précisément si c'est la fin du mot.

Formellement, montrons que, pour tout mot $w = aa^n bb^n$ avec $n \in \mathbb{N}^*$, l'automate accepte w. L'automate suit la séquence de configurations $(q_0, \gamma_0), (q_1, \gamma_0), (q_1, \gamma_0\alpha), \ldots, (q_1, \gamma_0\alpha^n), (q_2, \gamma_0\alpha^{n-1}), \ldots, (q_2, \gamma_0\alpha), (q_2, \gamma_0), (q_3, \gamma_0)$, et il accepte car q_3 est final.

À l'inverse, considérons un chemin de configurations, et montrons que le mot reconnu est de la forme $aa^n bb^n$ avec $n \in \mathbb{N}^*$. On commence à la configuration (q_0, γ_0), et on lit nécessairement un a pour parvenir à la configuration (q_1, γ_0). On lit alors un certain nombre de a, notons n ce nombre, et on parvient à la configuration $(q_1, \gamma_0\alpha^n)$. Il faut ensuite lire un b pour parvenir à la configuration $(q_1, \gamma_0\alpha^{n-1})$ ce qui impose que $n > 0$. Ensuite, pour parvenir à l'état final q_3, il faut que le sommet de pile soit γ_0, ce qui impose de lire $n - 1$ fois un b pour parvenir à la configuration (q_2, γ_0). Ensuite, il faut lire un autre b pour effectuer la transition et parvenir à (q_3, γ_0). On a alors lu $n + 1$ fois a, suivi de $1 + (n - 1) + 1$ fois b, avec $n > 0$, donc un mot de la forme attendue. Le mot se termine forcément à ce moment, car il n'y a plus de transition sortante. Ainsi, tout mot accepté par l'automate est de la bonne forme, ce qui conclut la preuve.

On en déduit qu'il existe des langages non rationnels reconnus par un automate à pile, car le langage $\{a^n b^n \mid n \geq 2\}$ n'est pas rationnel : ceci se montre sans difficulté par le lemme de l'étoile.

Question 2. On construit l'automate suivant sur l'alphabet de pile $\Gamma = \{\alpha, \gamma_0\}$:



Il est clair que tout mot avec autant de a que de b est accepté. En effet, considérons la fonction δ qui à toute position $0 \leq i \leq n$ d'un mot d'entrée w de longueur n associe $|w'|_a - |w'|_b$ pour w' le préfixe de longueur i du mot. On montre facilement par induction sur la position que, pour tout $0 \leq i \leq n$, si l'on considère la configuration où on se trouve après avoir lu le préfixe w' de longueur i de w :
— on a $\delta(i) = 0$ ssi la configuration est (q_0, γ_0)
— on a $\delta(i) > 0$ ssi la configuration est $(q_a, \gamma_0\alpha^{|\delta(i)|-1})$
— on a $\delta(i) < 0$ ssi la configuration est $(q_b, \gamma_0\alpha^{|\delta(i)|-1})$
Ainsi, le mot est accepté si et seulement si $\delta(|w|) = 0$, c'est-à-dire si et seulement s'il est dans le langage.

Question 3. Il est clair que $L_{\leq\eta}(A) \subseteq L(A)$ pour tout η, car tout calcul η-tronqué permet d'obtenir un calcul : la seule différence est qu'un calcul η-tronqué peut se bloquer parce que la pile est devenue vide, alors qu'elle n'aurait pas été vide dans un vrai calcul (i.e., on a dépilé plus profond que η).

Pour la seconde partie de la question, on construit simplement un automate fini sur les configurations η-tronquées : chaque configuration η-tronquée correspond à un état. Le point clé est que les configurations η-tronquées sont en nombre fini, donc on obtient bien un automate fini.

Question 4. Fixons w un mot de longueur n, l'automate à pile A, et $\eta \in \mathbb{N}^*$. Supposons que $w \in L(A) \setminus L_{\leq\eta}(A)$, et considérons un calcul acceptant $\chi = (q_0, z_0), \ldots, (q_p, z_p)$ de A sur w.

On observe d'abord que, clairement, χ est un calcul acceptant η-tronqué à moins qu'il existe un point $0 \leq m \leq p$ et un point $m < r' \leq p$ où $h_m - h_{r'} \geq \eta$. En effet, la seule façon que χ échoue en tant que calcul η-tronqué est qu'on atteigne la pile vide à un moment où la pile n'est pas véritablement vide, ce qui veut dire qu'on avait la pile à une hauteur h_m et qu'on atteint ensuite une hauteur de $h_m - \eta$ ou moins. En fait, vu que l'on ne peut dépiler les symboles que un par un, quitte à prendre $r' > m$ aussi petit que possible, on peut supposer que $h_m - h_{r'} = \eta$, et que r' est la plus petite valeur $> m$ pour laquelle c'est le cas. Ainsi, comme $w \notin L_{\leq\eta}(A)$, on sait qu'il existe un tel témoin $0 \leq m < r \leq p$ tels que $h_m - h_{r'} = \eta$ pour le calcul χ, et pour tout $m \leq j < m$, on a $h_j > h_{r'}$ par minimalité de r'.

Étant donné ce témoin, on cherche à construire une $(\eta - G)$-montagne. Comme la pile est initialement de hauteur 1, et qu'elle croît de G à chaque étape, il est clair qu'il existe $0 \leq l < m$ tel que $h_{r'} \leq h_l \leq h_{r'} + G$. On prend le plus grand $l < m$ tel que h_l ait cette propriété. On diminue ensuite r' en r de sorte à garantir que $h_r = h_l$ (ce qui assure par l'encadrement de h_l que $r' - r \leq G$) : ceci est possible car, vu que la taille de pile décroît de h_m à $h_{r'}$ et qu'on dépile au plus un symbole à chaque fois, toutes les valeurs intermédiaires sont forcément atteintes au cours de la descente (au contraire de ce qu'il se passait précédemment pour la montée). On prend le plus petit r avec cette propriété. On a maintenant construit $0 \leq l < m < r \leq p$ tels que $h_l = h_r$ et $h_r \geq \eta - G$. Il reste juste à observer que $h_j > h_l$ pour tout $l < j < r$, mais pour $j \leq m$ c'est une conséquence de la maximalité de l, et pour $m \leq j$ c'est une conséquence de la minimalité de r.

On a donc construit une $(\eta - G)$-montagne (l, m, r).

Question 5. On observe tout d'abord que, par les propriétés d'une montagne, le dernier symbole de la pile z_r est également γ, car ce symbole n'a pu être dépilé entre l et r.

Le nombre d'états de base est de $|Q|^2 \times |\Gamma|$, et on définit η_0 comme $|Q|^2 \times |\Gamma| \times (G+1)$, où G est défini à partir de A comme à la question précédente. Considérons un mot w et un calcul χ de A sur w qui a une η_0-montagne (l, m, r). De z_l à z_m, la pile passe d'une hauteur h_l à h_m ; comme on empile au plus G symboles à chaque étape de calcul, par définition de η_0, il y a au moins $v := 1 + |Q|^2 \times |\Gamma|$ hauteurs entre h_l et h_m qui sont atteintes à un indice entre l et m. Pour chacune de ces hauteurs, comme à la question précédente, il existe une position entre m et r où la même hauteur de pile est atteinte (toutes les hauteurs de pile sont atteintes pendant la descente). Ainsi, à partir de la η_0-montagne (l, m, r), on peut définir v montagnes imbriquées : spécifiquement, on a la séquence $h_l < h'_1 < \cdots < h'_v < h_m$ des hauteurs atteintes, et les indices $l < l'_1 < \cdots < l'_v < m$ où elles le sont, que l'on choisit chacun maximaux parmi les indices $< m$ qui atteignent la hauteur respective. On a ensuite la séquence des indices $m < r'_1 < \cdots < r'_v < r$ tels que $h_{l'_i} = h_{r'_i}$ pour tout $1 \leq i \leq v$, que l'on choisit chacun minimaux parmi les indices $> m$ qui atteignent la hauteur respective (il est clair qu'ils satisfont bien la relation d'ordre indiquée). Pour tout $1 \leq i \leq v$, il est ainsi clair que (l'_i, m, r'_i) est une η_i-montagne pour un certain $\eta_i > 0$: la seule condition à vérifier est celle sur les h_j, qui est vraie comme précédemment par maximalité des l'_i et minimalité des r'_i.

À présent, par définition de v et par le principe des tiroirs, il y a nécessairement deux de ces montagnes qui ont le même état de base, ce qui permet de conclure.

Question 6. On pose $\eta := \eta_0 + G$, pour η_0 comme défini à la question précédente. On sait par la question 3 que $L_{\leq\eta}(A)$ est un langage régulier : on pose p la longueur de pompage classique pour un automate A (sans pile) qui le reconnaît. Prenons un mot $w \in L$ avec $|w| > p$. Si $w \in L_{\leq\eta}(A)$, comme

$|w| > p$, on peut appliquer le pompage pour A et décomposer $w = uvz$ avec $|v| \geq 1$ (correspondant au cycle dans l'automate) et $uv^n z \in L$ pour tout $n \in \mathbb{N}$, et on conclut en posant $x = y = \epsilon$.

Si $w \in L(A) \setminus L_{\leq \eta}(A)$, alors on considère un calcul acceptant de A sur w, on note p sa longueur. D'après la question 4, ce calcul a une η_0-montagne. D'après la question 5, il existe deux montagnes (l', m, r') et (l'', m, r'') avec $l' < l'' < m < r'' < r'$ telles que $\beta(l', r') = \beta(l'', r'')$. On décompose $w = uvxyz$ suivant $0 \leq l' < l'' < r'' < r' \leq p$, ce qui garantit la première condition. Pour la seconde, on observe qu'on peut construire à partir de χ un calcul acceptant pour $uv^n xy^n z$ pour $n \in \mathbb{N}$. L'idée est que l'on peut reproduire les étapes du calcul correspondant à v et y : pour v, on part et on arrive au même état q_l avec le même sommet de pile qu'on peut lire mais qu'on ne peut jamais dépiler, et ce faisant on empile un certain mot z ; et pour y, on part et on arrive au même état q_r avec le même sommet de pile γ, en dépilant z sans descendre au-delà.

[La classe des langages reconnus par les automates à pile s'appelle également classe des langages algébriques*, et le résultat démontré à cette question est le pendant pour les automates à pile du théorème de Bar-Hillel, Perles et Shamir, cf [Wik17], avec un petit affaiblissement (on ne montre pas la condition $|vxz| \leq p$). La preuve proposée suit [AJ13].]*

Références

[AJ13] Antoine Amarilli and Marc Jeanmougin. A Proof of the Pumping Lemma for Context-Free Languages Through Pushdown Automata, 2013.

[Wik17] Wikipedia. Pumping lemma for context-free languages, 2017. `https://en.wikipedia.org/wiki/Pumping_lemma_for_context-free_languages`.