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. $A \wedge B \vdash A \longrightarrow B$
2. $A \vee B \vdash B \vee A$
3. $\vdash \neg(A \wedge \neg A)$

II Connecteur de Sheffer (d'après CCP 2020)

Remarque : cet exercice utilise uniquement de la logique propositionnelle (pas de démonstration sous forme d'arbre comme en déduction naturelle).

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 $\{x_1, x_2\}$.

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.
3. Exprimer ce connecteur en fonction de $\neg$ et $\wedge$.
4. Vérifier que $\neg x_1 = x_1 \diamond x_1$.
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.
6. Donner une forme normale conjonctive de la formule $x_1 \diamond x_2$.
7. Donner de même une forme normale disjonctive de la formule $x_1 \diamond x_2$.
8. Démontrer par induction sur les formules propositionnelles que l'ensemble de connecteurs $C = \{\diamond\}$ 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.

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.

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

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.

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 :

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

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 $m = aubv$.

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.

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).
7. Écrire une fonction `arbre_a_mot` de type `arbre -> mot` faisant l'inverse.
8. Montrer que le language L des mots de Dyck 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 $C(n) = \dfrac{1}{n+1}\dbinom{2n}{n}$.

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 :

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

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

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

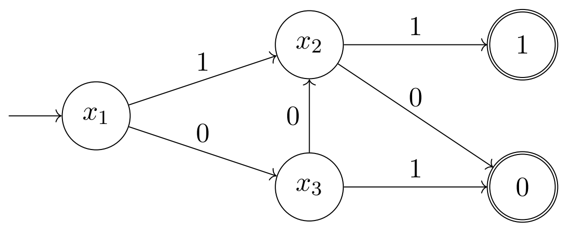
A1 – Profondeur de diagrammes de décision

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.

A9 – Automates pour les valuations de formules booléennes

Soit $\mathcal{X}$ un ensemble fini de variables. Une *valuation* de $\mathcal{X}$ est une fonction de $\mathcal{X}$ dans $\{0, 1\}$. Soit Φ une formule de la logique propositionnelle sur $\mathcal{X}$. On dit que la valuation ν *satisfait* Φ si la formule Φ s'évalue à 1 lorsque l'on remplace chaque variable dans Φ par son image par ν.

On fixe l'alphabet $\Sigma = \{0, 1\}$. Soit $<$ un ordre total sur $\mathcal{X}$: on écrira en conséquence $\mathcal{X} = x_1, \ldots, x_n$, avec $x_i < x_j$ pour tout $1 \leq i < j \leq n$. Le *mot suivant* $<$ d'une valuation ν de $\mathcal{X}$ est le mot $\nu(x_1) \cdots \nu(x_n)$ de longueur n sur l'alphabet Σ. Un *automate de valuations* pour Φ et $<$ est un automate A sur l'alphabet Σ tel que, pour tout mot $w \in \Sigma^*$, l'automate A accepte w si et seulement si $|w| = n$ et w est le mot suivant $<$ d'une valuation ν qui satisfait Φ.

Question 0. Construire un automate de valuations pour la formule $(x_1 \wedge x_3) \vee x_2$.

Question 1. Proposer un algorithme naïf qui, étant donné une formule Φ de la logique propositionnelle, construit un automate de valuations pour Φ. Discuter de la complexité de cet algorithme.

Dans les deux prochaines questions, on cherche à améliorer l'efficacité de cet algorithme.

Question 2. Pour tout $n \in \mathbb{N}$, on appelle L_n le langage sur Σ défini par $L_n := \{ww \mid w \in \Sigma^n\}$. Montrer que, pour tout $n \in \mathbb{N}$, pour tout automate A qui reconnaît L_n, l'automate A a au moins 2^n états.

Question 3. En utilisant la question précédente, montrer que, pour tout ensemble de variables totalement ordonné de taille paire $\mathcal{X} = x_1, \ldots, x_{2n}$, on peut construire une formule Φ_{2n} de taille $O(n)$ telle que tout automate de valuations pour Φ_{2n} et $<$ ait au moins 2^n états.

Qu'en déduire quant à l'algorithme de la question 1 ?

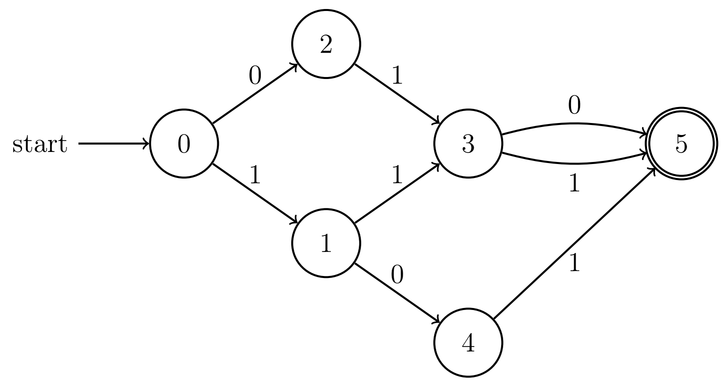
Suite des questions

Question 4. Montrer le même résultat qu'à la question précédente pour une famille de formules Φ'_{2n} qui utilise seulement les opérateurs $\vee$ et $\wedge$ (et pas $\neg$).

Question 5. Soit $\mathcal{X}$ un ensemble de variables arbitraire de taille paire totalement ordonné par $<$, soit $2n$ la taille de cet ensemble, et soit Φ_{2n} la formule définie à la question 3. Existe-t-il un ordre différent $<'$ sur $\mathcal{X}$ pour lequel il y ait un automate de valuations pour Φ_{2n} et $<'$ de taille plus faible que 2^n ?

Corrigé

Question 0. On construit par exemple l'automate suivant :



Question 1. On peut énumérer l'ensemble des valuations de $\mathcal{X}$. Pour chaque valuation ν, on teste si ν satisfait Φ, et on construit ainsi l'ensemble des valuations qui satisfont Φ. On construit explicitement l'ensemble L des mots suivant $<$ de ces valuations, et, comme ce langage est fini, on peut construire efficacement un automate qui reconnaît exactement ce langage : pour chaque mot de L, il y a un chemin d'états depuis l'état initial qui mène à un état final avec des transitions dont la concaténation des étiquettes forme w.

Si l'on suppose ici que chaque variable de $\mathcal{X}$ apparaît dans Φ, la complexité de cet algorithme est en $O(|\Phi| \times 2^n)$, car on fait pour chaque valuation une opération linéaire en la formule, et (dans l'automate) linéaire en n, or $n \leq |\Phi|$. En général, la complexité est en $O(\max(|\Phi|, n) \times 2^n)$.

Question 2. *Indication possible : proposer de traiter d'abord le cas d'un automate déterministe, pour lequel la preuve est un peu plus simple : il suffit de considérer l'image de tous les mots de Σ^n et de montrer que c'est une fonction totale et injective.*

Soit A un automate qui reconnaît L_n. On appelle Q l'ensemble d'états de A. Montrons que $|Q| \geq 2^n$.

Soit f la fonction de Σ^n dans 2^Q qui, a un mot $w \in \Sigma^n$, associe un état q_w tel qu'il existe un calcul de l'automate pour ww (c'est-à-dire un chemin de q_0 à un état final dont les transitions sont étiquetées par des lettres dont la concaténation forme ww) tel que q_w est l'état auquel on aboutit après avoir suivi n transitions. Notons qu'un tel calcul existe nécessairement, car ww est dans L_n et accepté par l'automate ; ainsi, la fonction f est une fonction totale.

Montrons à présent que f est injective. Procédons par l'absurde, et supposons qu'il existe $w_1 \neq w_2$ dans Σ^n tels que $f(w_1) = f(w_2)$. Soit $q := f(w_1)$. Ainsi, il existe un chemin π_1 dans l'automate étiqueté par w_1 qui va de q_0 à q, et un chemin π'_1 dans l'automate étiqueté par w_1 qui va de q à un état final q_1 ; et un chemin π_2 étiqueté par w_2 qui va de q_0 à q, et un chemin π'_2 étiqueté par w_2 qui va de q à un état final q_2. Considérons à présent le chemin formé en concaténant π_1 et π'_2 : ce chemin est étiqueté par

w_1w_2, et va de l'état initial q_0 à l'état final q_2, donc il montre que l'automate accepte w_1w_2. Or on a $w_1w_2 \in \Sigma^n$ mais $w_1 \neq w_2$, donc $w_1w_2 \notin L_n$, ce qui contredit le fait que A reconnaisse L_n. Ainsi, f est injective.

Comme $|\Sigma^n| = 2^n$, on déduit de l'existence de f que $|Q| \geq 2^n$, ce qu'il fallait démontrer.

Question 3. Fixons $n \in \mathbb{N}$. Le cas $n = 0$ est trivial avec la formule tautologique, donc on suppose $n > 0$.

Pour $1 \leq i \leq n$, soit Ψ_i la formule $(x_i \wedge x_{n+i}) \vee (\neg x_i \wedge \neg x_{n+i})$. Intuitivement, une valuation ν satisfait Ψ_i si et seulement si $\nu(x_i) = \nu(x_{n+i})$. Construisons la formule suivante :

$$\Phi_{2n} := \bigwedge_{1 \leq i \leq n} \Psi_i$$

Ainsi, une valuation ν de $\mathcal{X} := x_1, \ldots, x_{2n}$ satisfait Φ_{2n} si et seulement si, pour tout $1 \leq i \leq n$, on a $\nu(x_i) = \nu(x_{n+i})$. Ceci est le cas si et seulement si le mot de ν suivant $<$ est dans le langage L_n de la question précédente. Ainsi, d'après la question précédente, tout automate de valuations pour Φ_{2n} et $<$ a au moins 2^n états.

Or, la taille de Φ_{2n} est bien en $O(n)$, car chaque Ψ_i est de taille constante. Ceci conclut la preuve.

Ainsi, on ne peut pas espérer obtenir un algorithme efficace pour le problème de la question 1, parce que, dans le cas de la famille Φ_{2n}, un algorithme qui répond à ce problème doit matérialiser un automate de taille exponentielle en la formule d'entrée.

Question 4. On fixe à nouveau $n \in \mathbb{N}^*$. On définit la formule $\Psi'_i := x_i \vee x_{n+i}$, et on définit :

$$\Phi'_{2n} := \bigwedge_{1 \leq i \leq n} \Psi'_i$$

Cette formule utilise les bons opérateurs, et est toujours de taille $O(n)$. Montrons que tout automate de valuations pour Φ_{2n} et $<$ a 2^n états. Il est clair qu'une valuation ν de $\mathcal{X}$ satisfait Φ_{2n} si et seulement si le mot suivant $<$ de ν est dans le langage $L'_n := \{ww' \mid w, w' \in \Sigma^n \wedge \forall i \in \{1, \ldots, n\}, w_i = 1 \vee w'_i = 1\}$. Ainsi, montrons le pendant de la question 2 pour ce langage.

Soit un automate A qui reconnaisse L'_n, et soit Q son ensemble d'états. Montrons que $|Q| \geq 2^n$. Soit g la fonction de Σ^n dans Q qui, pour tout $w \in \Sigma^n$, considère un calcul de l'automate étiqueté par $w\overline{w}$, où $\overline{w}$ est le complément de w (c'est-à-dire le mot sur Σ tel que $w_i \neq \overline{w}_i$ pour tout $1 \leq i \leq n$). Ce mot est nécessairement dans L'_n, donc un tel calcul existe, et $g(w)$ choisit et renvoie un état q auquel on peut arriver après n transitions dans un tel calcul. Montrons que g est injective.

Procédons par l'absurde et supposons qu'il existe $w \neq w'$ dans Σ^n tel que $g(w) = g(w')$; soit $q := g(w)$. Comme $w \neq w'$, il existe $1 \leq i \leq n$ tel que $w_i \neq w'_i$; comme $\Sigma = \{0, 1\}$, quitte à échanger w et w', on peut supposer que $w_i = 1$ et $w'_i = 0$. Ainsi, $\overline{w}_i = 0$ mais $\overline{w'}_i = 1$. On considère à présent le chemin de l'état initial q_0 à q étiqueté par w', et le chemin de q à un état final étiqueté par $\overline{w}$. Ce chemin montre que $w'' := w'\overline{w}$ est accepté par l'automate. Pourtant, on sait que $w''_i = 0$ et $w''_{n+i} = \overline{w}_i = 0$. Ainsi $w'' \notin L'_n$. On est donc parvenu à une contradiction.

Question 5. Fixons $n \in \mathbb{N}$, et $\mathcal{X}$. Considérons l'ordre $<'$ défini comme suit :

$$x_1 <' x_{n+1} <' x_2 <' x_{n+2} <' \cdots <' x_{n-1} <' x_{2n-1} <' x_n <' x_{2n}.$$

Ainsi, l'ordre suivant $<'$ des valuations qui satisfont Φ_{2n} est à présent $(00|11)^n$.

Construisons un automate de valuations pour Φ_{2n} et $<'$ de taille $O(n)$. On définit l'ensemble d'états comme $Q := \{q_0, q_1, \ldots, q_n\} \cup \{q_i^b \mid 1 \leq i \leq n, b \in \{0, 1\}\}$, l'état initial est q_0, l'état final est q_n, et les transitions sont comme suit : pour $1 \leq i \leq n$, pour $b \in \{0, 1\}$, une transition étiquetée b de q_{i-1} à q_i^b et de q_i^b à q_i.

Il est clair que l'automate est bien de taille linéaire. On montre par récurrence descendante pour i de n à 0 que le langage reconnu à partir de l'état q_i est $(00|11)^{n-i}$. Le cas de base est celui de l'état q_n, qui est final et n'a pas de transition sortante, donc reconnaît le langage $\{\epsilon\}$. Pour la récurrence, si l'on suppose le résultat pour q_i pour un certain $0 < i \leq n$, on montre le résultat pour q_{i-1} car les mots acceptés depuis q_{i-1} sont les mots de q_i précédés de 00 ou de 11. Ceci prouve que l'automate obtenu reconnaît effectivement le bon langage.