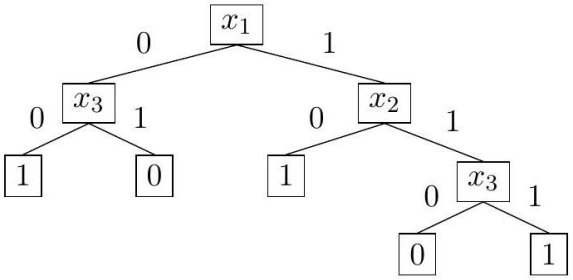
I Arbre de décision

On considère un ensemble de variables propositionnelles $\mathcal{X} = \{x_1, \ldots, x_n\}$ muni de l'ordre total où $x_i < x_j$ si et seulement si $i < j$. Un arbre de décision sur $\mathcal{X}$ est un arbre binaire dont les feuilles sont étiquetées par 0 ou par 1 , et dont les nœuds internes sont étiquetés par une variable de $\mathcal{X}$ et ont deux enfants appelés enfant 0 et enfant 1 . On impose que si un nœud interne n étiqueté par x_i a un descendant n' qui est un nœud interne étiqueté par x_j alors $x_i < x_j$.

Un arbre de décision T sur $\mathcal{X}$ décrit une fonction booléenne Φ_T qui à toute valuation $\nu : \mathcal{X} \to \{0, 1\}$ associe une valeur de vérité calculée comme suit : si T consiste exclusivement d'une feuille étiquetée $b \in \{0, 1\}$ alors la fonction Φ_T s'évalue à b quelle que soit ν. Sinon, on considère le nœud racine n de T et la variable x_i qui l'étiquette, on regarde la valeur $\nu(x_i) \in \{0, 1\}$ que ν donne à x_i, et le résultat de l'évaluation de Φ_T sous ν est celui de l'évaluation de $\Phi_{T'}$ sous ν, où T' est le sous-arbre de T enraciné en l'enfant b de n.

1. On considère l'arbre de décision T_0 suivant et la fonction Φ_{T_0} qu'il définit. Évaluer cette fonction pour la valuation donnant à x_1, x_2, x_3 respectivement les valeurs $0, 1, 0$. Donner un exemple de valuation sous laquelle cette formule s'évalue en 0 .



2. On considère la formule de la logique propositionnelle $(x_1 \wedge x_2) \vee \neg(x_1 \wedge \neg x_3)$. Construire un arbre de décision sur les variables $x_1 < x_2 < x_3$ qui représente la même fonction.
3. Quels arbres de décision représentent des tautologies? des fonctions satisfiables?
4. Étant donné un arbre de décision représentant une fonction Φ, expliquer comment construire un arbre de décision représentant sa négation $\neg\Phi$.
5. Étant donné un arbre de décision représentant une fonction Φ, donner le pseudocode d'un algorithme qui calcule une représentation de Φ sous la forme d'une formule de la logique propositionnelle. Analyser sa complexité en temps et en espace.
6. Étant donné deux arbres de décision représentant des formules Φ_1 et Φ_2 sur le même ensemble de variables et avec le même ordre, expliquer comment construire un arbre de décision représentant $\Phi_1 \wedge \Phi_2$. Quelle est sa complexité en temps et la taille de l'arbre ainsi obtenu?
7. Étant donné un arbre de décision et la séquence de variables $x_1, \ldots, x_n$, donner le pseudocode d'un algorithme qui calcule combien de valuations satisfont la fonction booléenne qu'il capture. Quelle est sa complexité en temps?
8. Peut-on efficacement récrire une formule quelconque de la logique propositionnelle en arbre de décision?

   On se propose dans les quelques prochaines questions de démontrer formellement l'intuition de la question 7.

7a. Une formule booléenne en forme normale disjonctive, également appelée DNF, est une formule qui est une disjonction de conjonctions de littéraux (c'est-à-dire une variable ou sa négation).

   Montrer qu'étant donné un arbre de décision on peut efficacement calculer une formule en DNF qui représente la même fonction.

7b. Montrer que, pour tout n, la formule $\bigwedge_{1 \leq i \leq n} (x_i \vee y_i)$ n'admet pas de représentation concise sous la forme d'une DNF.

7c. Conclure.

II 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.

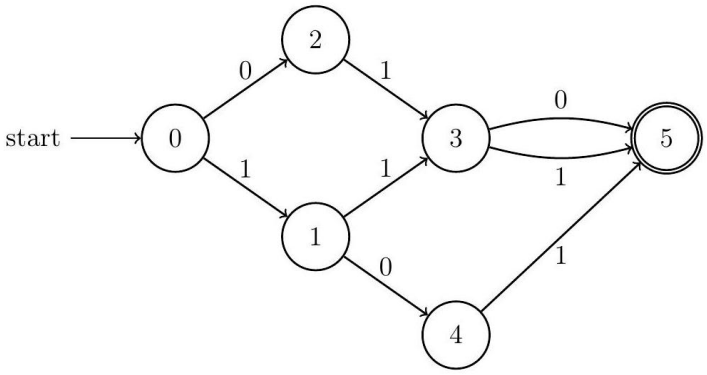
1. Construire un automate de valuations pour la formule $(x_1 \wedge x_3) \vee x_2$.
2. Proposer un algorithme naïf qui, étant donné une formule Φ de la logique propositionnelle, construit un automate de valuations pour Φ. Discuter de la complexité de cet algorithme.

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

3. 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 reconnait L_n, l'automate A a au moins 2^n états.
4. 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 ?
5. 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 $\neg$).
6. 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 ?

1. On construit par exemple l'automate suivant:



2. 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 reconnait 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 $\mathcal{X}$ apparait 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 $O(\max(|\Phi|, n) \times 2^n)$.

3. 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.

4. 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.

5. 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.

6. 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 $(00 \mid 11)^n$.

Construisons un automate de valuations pour Φ_{2n} et $<'$ de taille $O(n)$. On définit l'ensemble d'états comme $Q := \{q_0, q_1, \ldots, q_n\} \cup \left\{q_i^b \mid 1 \leq i \leq n, b \in \{0, 1\}\right\}$, 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 $(00 \mid 11)^{n-i}$. Le cas de base est celui de l'état q_n, qui est final et n'a pas de transition sortante, donc reconnait le langage $\{\varepsilon\}$. 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 reconnait effectivement le bon langage.

III Typage OCaml

On souhaite formaliser le typage OCaml. On notera $\Gamma \vdash e : \tau$ si l'expression OCaml e est typée par le type τ, où Γ est un **environnement de typage**, c'est-à-dire une fonction donnant un type à chaque variable.

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash n : \texttt{int}}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \texttt{fun}\ x \texttt{ -> } e : \sigma \texttt{ -> } \tau} \qquad \frac{\Gamma \vdash f : \sigma \texttt{ -> } \tau \quad \Gamma \vdash e : \sigma}{\Gamma \vdash f\ e : \tau}$$

1. Soit $\Gamma = \{$`f : a -> (b -> a)`, `g : b -> a`$\}$. Montrer que `fun x -> f (g x) x` est bien typé.
2. Quelles analogies peut-on faire entre le typage OCaml et la déduction naturelle ?
3. Montrer que `(fun f -> f 1 2)(fun x -> 3)` n'est pas typable.

On ajoute maintenant les tuples :

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2}$$

On veut aussi ajouter des fonctions polymorphes.

4. En utilisant des quantificateurs, proposer des types pour `fst` et `snd`, et une règle d'élimination.
5. Montrer alors que `fst (42, true)` est bien typé.

IV Sujet 0 CCP MPI

Rappelons les règles de déduction naturelle suivantes, où A et B sont des formules logiques et Γ un ensemble de formules logiques quelconques :

$$\frac{}{\Gamma, A \vdash A}\ \text{AX} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_e \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_i \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash A \to B}{\Gamma \vdash B}\ \to_e \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg_i \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \neg_e$$

1. Montrer que le séquent $\vdash \neg A \to (A \to \bot)$ est dérivable, en explicitant un arbre de preuve.
2. Montrer que le séquent $\vdash (A \to \bot) \to \neg A$ est dérivable, en explicitant un arbre de preuve.
3. Donner une règle correspondant à l'introduction du symbole $\wedge$ ainsi que deux règles correspondant à l'élimination du symbole $\wedge$. Montrer que le séquent $\vdash (\neg A \to (A \to \bot)) \wedge ((A \to \bot) \to \neg A)$ est dérivable.
4. On considère la formule $P = ((A \to B) \to A) \to A$ appelée loi de Peirce. Montrer que $\models P$, c'est-à-dire que P est une tautologie.
5. Pour montrer que le séquent $\vdash P$ est dérivable, il est nécessaire d'utiliser la règle d'absurdité classique $\bot_c$ (ou une règle équivalente), ce que l'on fait ci-dessous (il n'y aura pas besoin de réutiliser cette règle). Terminer la dérivation du séquent $\vdash P$, dans laquelle on pose $\Gamma = \{(A \to B) \to A, \neg A\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{?}{\Gamma \vdash A}\ ? \qquad \dfrac{}{\Gamma \vdash \neg A}\ \text{AX}}{\Gamma = (A \to B) \to A, \neg A \vdash \bot}\ \neg_i}{(A \to B) \to A) \vdash A}\ \bot_c}{\vdash ((A \to B) \to A) \to A}\ \to_i$$