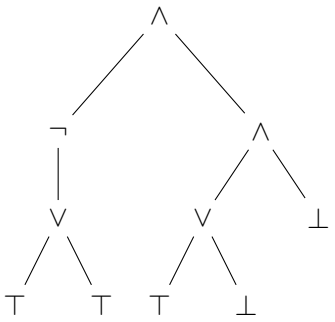
Exercice 1 (Représentation des termes):

Un terme booléen est un arbre binaire étiqueté par $\wedge, \vee, \neg, \top, \bot$, tel que un noeud étiqueté par $\wedge$ ou $\vee$ ait deux successeurs, un noeud étiqueté par $\neg$ ait un successeur et un noeud étiqueté par $\top$ ou $\bot$ ait zéro successeur.

Un terme booléen peut être représenté par un graphe étiqueté (le graphe étant représenté avec une liste d'adjacence), ou comme un mot sur l'alphabet $\Sigma = \{\wedge, \vee, \neg, \top, \bot, (,)\}$.



$\wedge(\neg(\vee(\top,\top)),\wedge(\vee(\top,\bot),\bot))$

1. Montrer qu'il existe deux fonctions logspace permettant de passer d'une représentation à l'autre.

   Solution:
   Pour passer d'une liste d'adjacence à un mot, il faut d'abord trouver la racine, en cherchant dans l'entrée le noeud qui n'apparait pas dans aucune des listes. C'est une boucle **for** parcourant l'ensemble des noeuds, un compteur $\leq n$ suffit (qui prend une place $\leq \log(n)$ sur la bande de travail). On effectue ensuite un parcours en profondeur en commençant toujours pas le fils gauche, et en stockant l'identifiant du noeud courant sur la bande de travail (toujours en espace $\leq \log(n)$). Lorsqu'on remonte, si on remonte depuis un fils gauche, on traite le fils suivant, sinon on continue de remonter.

   Pour passer de la représentation par un mot vers une liste d'adjacence, la bande de travail est utilisée lorsqu'on arrive à une feuille d'un terme, pour savoir jusqu'où on doit remonter. Cela peut se faire via un compteur qui compte le nombre de parenthèses ouvertes/fermées sur la bande d'entrée. La valeur du compteur est $\leq n$, donc en l'écrivant en binaire on a bien une bande de travail $\leq \log(n)$.

2. On peut naturellement définir la fonction eval qui évalue un terme booléen :

$$\begin{aligned}
\mathsf{eval}(\top) &= 1 \\
\mathsf{eval}(\bot) &= 0 \\
\mathsf{eval}(\neg(t)) &= (1 + \mathsf{eval}(t)) \mod 2 \\
\mathsf{eval}(\wedge(t_1, t_2)) &= \mathsf{eval}(t_1).\mathsf{eval}(t_2) \\
\mathsf{eval}(\vee(t_1, t_2)) &= 1 - (1 - \mathsf{eval}(t_1)).(1 - \mathsf{eval}(t_2))
\end{aligned}$$

   Montrer que le problème suivant est dans L :
   Donnée : Le codage en liste d'adjacence d'un terme booléen t.
   Question : $\mathsf{eval}(t) = 1$?

   Solution:
   On parcourt l'arbre en profondeur, en évaluant le fils gauche d'abord, et en n'évaluant le fils droit que si nécessaire : une fois arrivé sur une feuille, on remonte.

   - Si le père est un $\wedge$:

- Si la valeur courante est 0, on continue de remonter avec la valeur courante 0.
- Si la valeur courante est 1 et qu'on remonte depuis le fils gauche, on évalue le fils droit.
- Si la valeur courante est 1 et qu'on remonte depuis le fils droit, cela veut dire qu'on avait évalué le fils gauche à 1, et donc on peut remonter avec la valeur courante 1.

- Le cas où le père est un $\vee$ est analogue : on n'évalue le fils droit que si le fils gauche a été évalué à 0.

Exercice 2 (Compression linéaire):

Montrer que pour toute fonction f et tout entier $k > 0$ on a les inclusions suivantes :

- $\text{SPACE}(kf) \subseteq \text{SPACE}(f)$
- $\text{NSPACE}(kf) \subseteq \text{NSPACE}(f)$

Solution:

Étant donnée un machine M travaillant sur des caractères de Σ, on construit une machine M' qui travaille comme M mais sur des blocs de k caractères autrement dit sur l'alphabet Σ^k.

La position précise de la tête de lecture dans un bloc se code dans les états.

Exercice 3 (Accélération linéaire):

Montrer que $\forall c > 0$, $\text{TIME}(f(n)) \subseteq \text{TIME}(cf(n) + n + 2)$.

Évaluer le nombre d'états et de transitions de la nouvelle machine de Turing.

Solution:

Soit k tel que $1/k \leq c$. Supposons qu'une machine M décide un langage L en $f(n)$ étapes, avec r symboles de ruban et s états. On construit une machine M' avec r^{3k} symboles de ruban, chaque symbole représentant un bloc de $3k$ symboles du ruban de M. Le ruban de M' est un représentation compressée du ruban de M, avec la cellule i qui représente les $2i-1$-ème, $2i$-ème et $2i+1$-ème blocs de k lettres du ruban de M (ces blocs se chevauchent). En une étape de calcul, M' simule le calcul de M jusqu'à ce que la tête de lecture de M quitte le bloc courant de taille $3k$ par la droite ou la gauche (cette simulation est possible en une étape car M ne peut pas être dans plus de $3sr^{3k}$ configurations possibles sans quitter le bloc). Pendant cette simulation, si M accepte alors M' accepte, si M boucle alors M' ne change pas de configuration (et donc boucle aussi).

Après modification du bloc lu et déplacement de la tête de lecture, une partie du bloc sous la nouvelle position de la tête de lecture n'est plus correcte, mais la bonne valeur (de taille bornée) peut être codée dans l'état. Il faut donc $2r^k$ copies de chaque état (codage de la partie fausse du bloc, et information sur le fait qu'on vient de la droite ou de la gauche).

Chaque étape de M' correspond à au moins k étapes de M, donc M' termine en $f(n)/k$ étapes après un nombre linéaire d'étapes initiales pour convertir l'entrée dans la représentation compressée.

Exercice 4 (3SAT):

Un *littéral* est soit une proposition atomique soit la négation d'une proposition atomique. Une *clause* (disjonctive) est une disjonction de littéraux. Une formule sous forme normale conjonctive (CNF) est une conjonction de clauses. Ainsi $\varphi \equiv p_1 \wedge (p_2 \vee \neg p_1)$ est une formule CNF composée de deux clauses. Une formule 3CNF est une formule CNF dont les clauses ont au plus trois littéraux. 3SAT est le problème correspondant à la satisfiabilité d'une formule 3CNF.

Deux formules φ et ψ sont équivalentes si pour toute interprétation ν, on a : $\nu(\varphi) = \nu(\psi)$. Pour toute formule, on peut construire une formule CNF équivalente comme suit.

- Pousser les négations devant les propositions :
  - $\neg\neg\varphi \equiv \varphi$;
  - $\neg(\varphi_1 \vee \varphi_2) \equiv (\neg\varphi_1) \wedge (\neg\varphi_2)$;
  - $\neg(\varphi_1 \wedge \varphi_2) \equiv (\neg\varphi_1) \vee (\neg\varphi_2)$.
- Pousser les disjonctions sous les conjonctions : $(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \equiv (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$.

Cependant la formule obtenue est de taille exponentielle et cette explosion est inévitable.

Montrer que 3SAT est NP-difficile (donc NP-complet).

Solution:

Soit φ une formule arbitraire, on construit en temps polynomial une formule 3CNF ψ comme suit.

- Pour toute occurrence d'un opérateur, on ajoute une nouvelle proposition et on étiquette le noeud de l'arbre syntaxique de la formule par cette proposition ;
- Les clauses of ψ sont définies comme suit. La proposition étiquetant la racine est une clause et pour tout sommet interne de l'arbre :
  - Si c'est une négation étiquetée par x et y étiquette son fils, alors $\neg x \vee \neg y$ et $x \vee y$ sont des clauses ;
  - Si c'est une conjonction étiquetée par x et y, z étiquettent ses fils, alors $x \vee \neg y \vee \neg z$ et $\neg x \vee y$, $\neg x \vee z$ sont des clauses ;
  - Si c'est une disjonction étiquetée par x et y, z étiquettent ses fils, alors $\neg x \vee y \vee z$ et $x \vee \neg y$, $x \vee \neg z$ sont des clauses.

Voici un exemple de traduction.

$$\begin{aligned}\varphi &= (\neg(p \wedge q)) \vee r \\ \psi &= x_v \wedge (\neg x_v \vee x_n \vee r) \wedge (x_v \vee \neg x_n) \wedge (x_v \vee \neg r) \\ &\quad \wedge\ (\neg x_n \vee \neg x_w) \wedge (x_n \vee x_w) \\ &\quad \wedge (x_w \vee \neg p \vee \neg q) \wedge (\neg x_w \vee p) \wedge (\neg x_w \vee q)\end{aligned}$$

Montrons que φ est satisfaisable si et seulement si ψ est satisfaisable.

- Supposons que $\nu \models \varphi$. Définissons ν' étendant ν sur les nouvelles propositions ainsi. Soit x une proposition correspondant à un noeud interne de sous-formule φ_x. On choisit $\nu'(x) = \nu(\varphi_x)$. On vérifie immédiatement que $\nu' \models \psi$.
- Supposons que $\nu \models \psi$. Par induction sur la taille des sous-formules, on établit que $\nu(x) = \nu(\varphi_x)$. En examinant la clause x où x étiquette la racine, on obtient $\nu(\varphi) = \top$.

Exercice 5 (1-IN-3SAT):

1-IN-3SAT est le problème suivant :

Donnée : Un ensemble des clauses $C_1, \cdots, C_m$, $m > 1$; chaque C_i est une disjonction d'exactement 3 littéraux.

Question : Existe-t-il une valuation telle que seulement un littéral soit vrai dans chaque C_i ?

Montrer que 1-IN-3SAT est NP-difficile (et donc NP-complet).

Solution:

On effectue une réduction polynomiale de 3SAT à 1-IN-3SAT.

Soit $C_i = \{x_{i_1}, x_{i_2}, x_{i_3}\}$ une clause d'une instance de 3SAT. Dans l'instance de 1-IN-3SAT correspondante, on produit les trois clauses suivantes, avec quatre variables fraiches a_i, b_i, c_i, d_i :

$$C_{i_1} = \{\overline{x_{i_1}}, a_i, b_i\}, C_{i_2} = \{x_{i_2}, b_i, c_i\}, C_{i_3} = \{\overline{x_{i_3}}, c_i, d_i\}$$

Ainsi, si l'instance de 3SAT a n variables et m clauses, l'instance 1-IN-3SAT correspondante aura $n + 4m$ variables et $3m$ clauses. Cette transformation peut être faite en temps polynomial. Montrons que l'instance initiale est acceptée par 3SAT si et seulement si sa traduction est acceptée par 1-IN-3SAT.

Supposons que l'instance de 3SAT n'est pas satisfaisable. Alors, pour toute valuation, il existe une clause C_i où on a $x_{i_1} = \bot$, $x_{i_2} = \bot$ and $x_{i_3} = \bot$. Essayons d'étendre une telle valuation afin d'avoir une instance valide pour les clauses $C_{i_1}, C_{i_2}, C_{i_3}$: Dans C_{i_2}, comme $x_{i_2} = \bot$, exactement b_i ou c_i doit être vrai ; si $b_i = \top$, C_{i_1} ne peut pas être satisfait ; si $c_i = \top$, C_{i_3} ne peut pas être satisfait. Il est donc impossible de satisfaire exactement un littéral de chaque clause de l'instance de 1-IN-3SAT correspondante.

Réciproquement, supposons que l'instance de 3SAT est satisfaisable. Il existe donc une valuation telle que chaque C_i est vrai. Montrons qu'on peut toujours étendre une telle valuation pour satisfaire l'instance de 1-IN-3SAT correspondante :

- Si $x_{i_2} = \top$, alors on doit poser $b_i = \bot$ et $c_i = \bot$ pour que C_{i_2} soit satisfait. On peut alors poser $a_i = x_{i_1}$ et $d_i = x_{i_3}$ pour que C_{i_1} et C_{i_3} soient correctement satisfaits.
- Si $x_{i_2} = \bot$:
  - Si $x_{i_1} = \top$ et $x_{i_3} = \top$, on pose $a_i = \top$, $b_i = \bot$, $c_i = \top$, $d_i = \bot$ et les trois clauses C_{i_k} sont correctement satisfaites.
  - Si seulement $x_{i_1} = \top$, on pose $a_i = \bot$, $b_i = \top$, $c_i = \bot$, $d_i = \bot$ et les trois clauses C_{i_k} sont correctement satisfaites.
  - Si seulement $x_{i_3} = \top$, on pose $a_i = \bot$, $b_i = \bot$, $c_i = \top$, $d_i = \bot$ et les trois clauses C_{i_k} sont correctement satisfaites.

Au final, une valuation rendant vraie l'instance de 3SAT peut toujours être étendue en une valuation satisfaisant correctement son instance de 1-IN-3SAT correspondante.

Donc 1-IN-3SAT est NP-difficile.