La puissance de l'espace : de la connectivité aux jeux

Adam Shimi

2 avril 2019

Plan

1. Différences entre l'espace et le temps
   - Détails préliminaires
   - Classes élémentaires
2. Moins que l'entrée : espace logarithmique
   - Calcul en espace logarithmique
   - Vérification en espace logarithmique
3. Surpasser tous les joueurs : espace polynomial

Pourquoi ne pas se contenter du temps ?

L'intérêt de l'espace

- L'espace (la mémoire) est aussi une ressource précieuse dans la plupart des contextes, parfois plus que le temps.
  Par exemple, elle est cruciale dans les systèmes embarqués.
- Même si un algorithme est rapide, il faut que sa consommation en espace n'explose pas.
- L'espace se comporte fondamentalement différement du temps : il est réutilisable.
- Il n'y a pas de sous-classes intéressantes de $\mathcal{P}$ définies par le temps, mais il y a plusieurs candidats définis par l'espace.

Subtilités sur l'espace (1/2)

Qu'est-ce qu'on mesure ?

Qu'est-ce que la complexité en espace ?
Uniquement l'espace utilisé par le calcul.
C'est-à-dire qu'on ne compte pas l'entrée ni la sortie dans l'espace utilisé. Donc uniquement la partie de la bande de travail utilisée pour traiter l'entrée.

Mais on court alors le risque de donner de l'espace "gratuit", si l'on peut par exemple réutiliser l'espace pour l'entrée ou celui pour la sortie.
La solution :

- L'espace d'entrée doit être uniquement en lecture.
- L'espace de sortie doit servir uniquement en écriture.

Subtilités sur l'espace (2/2)

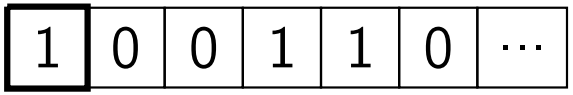
Définition de l'espace

La définition standard calcule la complexité en espace comme le nombre de cases mémoires utilisées.
Nous prendrons plutôt le nombre de bit utilisés par l'algorithme.
C'est-à-dire le nombre de bit nécessaire pour encoder les symboles de l'alphabet multiplié par le nombre de cases utilisées.

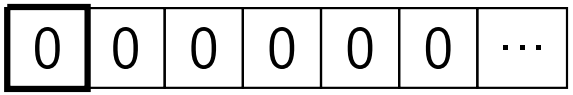
Plusieurs avantages :

- Relativement indépendant du modèle, et notamment de l'alphabet.
- On peut y rajouter facilement le nombre d'états, pour prendre en compte toute la capacité de travail.
- Cela simplifie les preuves.

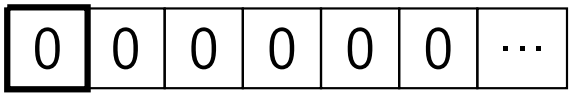
Modèle standard



Lecture seule



Lecture/Écriture



Écriture seule

Notre modèle

Même si nos définitions et nos résultats sont en général indépendants du modèle, il est plus aisé de réfléchir sur une architecture particulière pour l'espace.
Nous utiliserons dont une machine de Turing à 3 bandes :

- Une bande d'entrée en lecture seule.
- Une bande de travail classique, en lecture et écriture.
- Une bande de sortie en écriture seule.

Graphe de configuration

Définition

Le graphe de configuration d'une machine de Turing M sur l'entrée x en espace $O(S(|x|))$ est le graphe tel que :

- Les noeuds sont les configurations de M avec entrée x, c'est-à-dire l'état, le contenu de la bande de travail et les positions des têtes de lecture et de travail.
- Les arêtes (dirigées) relient une configuration à une autre suivante d'après la fonction de transition de M.

Pas besoin de s'occuper de la bande de sortie, puisqu'elle est en écriture seule, et que dans un problème de décision il n'y a qu'un bit à écrire, à la fin.

Propriétés du graphe de configuration (1/2)

Nombre de noeuds

Si $log(n) = O(S(n))$, le graphe de configuration a $2^{O(S(n))}$ noeuds.

- Il y au plus autant de configurations que le produit du nombre d'états $|Q|$, du nombre de valeurs des $O(S(n))$ bits et du nombre de positions possibles des têtes.
- Par définition de M, il existe une constante $c \in \mathbb{R}^{+*}$ telle que le nombre de valeurs possibles des cases est $2^{cS(n)}$.
- La position de la tête de travail est sur l'une des $cS(n)$ cases, c'est-à-dire qu'elles prennent $log(cS(n))$ bits à spécifier. Et celle de la tête de lecture est sur une des n cases, et prend donc $log(n) = O(S(n))$ bits, c'est à dire $\leq dS(n)$.
- Le produit total nous donne $|Q| * log(cS(n)) * dS(n) * 2^{cS(n)}$. Comme $log(c(S(n))) * dS(n) = o(2^{cS(n)})$, il existe $\gamma \in \mathbb{R}^{+*}$ telle que $2^{\gamma S(n)}$ majore ce produit.

Propriétés du graphe de configuration (2/2)

Description d'une configuration

Chaque configuration se décrit avec $O(S(n))$ bits.

Comme il y a $2^{O(S(n))}$ états, il suffit de les numéroter et de garder le compteur en mémoire, ce qui prend $log(2^{O(S(n))}) = O(S(n))$ bits.

Existence d'une arête

On peut vérifier l'existence d'une arête entre deux noeuds en espace $O(S(n))$.

- L'existence de l'arête ne dépend que d'un bit de x, d'un bit de la bande de travail, des positions des têtes et des deux états.
- On énumère les transitions possibles, et pour chacune on vérifie que la configuration initiale satisfait les conditions de départ et la configuration suivante les conditions d'arrivée.

Classes de complexité : $\mathcal{DSPACE}$

Détail préliminaire : constructibilité en espace

Une fonction $f : \mathbb{N} \mapsto \mathbb{N}$ est **constructible en espace** $\triangleq \exists M$ une machine de Turing qui calcule $f(n)$ en espace $O(f(n))$.
Contrairement au temps, $log(n)$ est constructible en espace.
On se limitera aux fonctions constructibles en espace pour les bornes supérieures.

Définition

Soit f une fonction constructible en espace. Alors un langage $L \subseteq \{0,1\}^*$ appartient à la classe de complexité $\mathcal{DSPACE}(f(n)) \triangleq \exists M$ une machine de Turing :

- $\forall x \in \{0,1\}^*$: M utilise $O(f(n))$ bits pour décider x.
- $\forall x \in L$: M accepte x.
- $\forall x \notin L$: M rejette x.

Théorème de hiérarchie spatiale

Hiérarchie spatiale

Soit f, g constructibles en espace avec $f(n) = o(g(n))$. Alors $\mathcal{DSPACE}(f(n)) \subsetneq \mathcal{DSPACE}(g(n))$.

Raisonnement analogue au cas temporel. Mais le coût en espace d'une simulation est un produit par une constante : il faut juste $S(n)$ d'espace supplémentaire pour compter les pas de la machine simulée et $log(S(n))$ pour garder la position de la tête simulée en mémoire.

Différences avec le cas temporel

- Commence à $log(n)$ plutôt qu'à n
- Pas besoin du facteur $log(f(n))$

Plan

1. Différences entre l'espace et le temps
   - Détails préliminaires
   - Classes élémentaires
2. Moins que l'entrée : espace logarithmique
   - Calcul en espace logarithmique
   - Vérification en espace logarithmique
3. Surpasser tous les joueurs : espace polynomial

Pourquoi s'intéresser à l'espace logarithmique ?

Valeur de l'espace logarithmique

- $log(n)$ est constructible en espace, cela a donc du sens.
- Ensuite, il existe des problèmes fondamentaux qui se calculent ou se vérifient en espace logarithmique.
- Cela nous permet de définir des sous-classes non triviales de P, qui ont une forte chance d'être des sous-classes propres.
- Enfin, cela permet de définir les réductions en espace logarithmique, tout aussi importantes que celles en temps polynomial pour la théorie de la complexité.

La classe $\mathcal{L}$

Définition

La classe de complexité $\mathcal{L} \triangleq \mathcal{DSPACE}(log(n))$.

Le problème fondamental de $\mathcal{L}$

Déterminer la connectivité d'un graphe non orienté est dans $\mathcal{L}$. C'est la preuve qu'on peut résoudre des problèmes non triviaux en espace logarithmique.

Par contre, la preuve est une des plus grandes réussites de la théorie de la complexité des 20 dernières années, et est donc bien trop technique pour tenir dans ce cours.

La classe $\mathcal{NL}$

Définition

Un language $L \subseteq \{0,1\}^*$ appartient à la classe de complexité $\mathcal{NL} \triangleq \exists M$ une machine de Turing en espace logarithmique, $\exists p$ un polynome :

- $\forall x \in L, \exists y \in \{0,1\}^{p(|x|)}$: M accepte $\langle x, y \rangle$.
- $\forall x \notin L, \forall y \in \{0,1\}^*$: M rejette $\langle x, y \rangle$.

Il existe deux définitions possibles pour la vérification en espace logarithmique, selon la façon dont le certificat est fourni :

- Si le certificat est fourni sur une bande où l'on peut lire plusieurs fois, on obtient $\mathcal{NL}_{offline}$.
- Si le certificat est fourni sur une bande à usage unique – c'est-à-dire où la tête ne bouge que vers la droite –, on obtient $\mathcal{NL}_{online}$.

Quelle définition de $\mathcal{NL}$ choisir ?

Problèmes avec $\mathcal{NL}_{offline}$

- $\mathcal{NL}_{offline} = \mathcal{NP}$. En effet, si l'on a une valuation, il est possible d'évaluer une instance de 3SAT en espace logarithmique en évaluant clause par clause.
- $\mathcal{NL}_{offline}$ "triche", puisque l'espace utilisé pour "internaliser" la preuve n'est pas pris en compte (il y a toujours moyen de relire).
- $\mathcal{NL}_{offline}$ ne correspond pas à la classe de complexité des problèmes calculables par une machine non-déterministe en espace logarithmique, la définition originale de $\mathcal{NL}$.

Pour ces raisons, on définit $\mathcal{NL} = \mathcal{NL}_{online}$.

Le problème le plus important dans $\mathcal{NL}$: st-CONN

Définition

Le problème de décision st-CONN est l'ensemble des triplets $\langle G, s, t \rangle$ où G est un graphe orienté, s et t sont deux noeuds de G, et G contient un chemin reliant s à t.

Brique de base de beaucoup de problèmes de décision, voire même de beaucoup de problèmes pratiques.

st-CONN est $\mathcal{NL}$-complet

D'ailleurs, st-CONN est complet pour $\mathcal{NL}$.

Complétude dans $\mathcal{NL}$: réductions en espace logarithmique

Les réductions en temps polynomial peuvent utiliser un espace polynomial ; il faut donc une nouvelle forme de réduction.

Fonction implicitement calculable en espace logarithmique

Une fonction implicitement calculable en espace logarithmique est une fonction $f : \{0,1\}^* \mapsto \{0,1\}^*$ telle que $\exists M$ une machine de Turing en espace logarithmique qui prend une entrée $x \in \{0,1\}^*$ et un entier i (codé en binaire), et retourne le i-ème bit de $f(x)$.

Réductions en espace logarithmique

Soit L et L' deux langage dans $\{0,1\}^*$. On dit que L est réductible en espace logarithmique à L' s'il existe une fonction f implicitement calculable en espace logarithmique telle que
$\forall x \in \{0,1\}^* : x \in L \iff f(x) \in L'$.

st-CONN est $\mathcal{NL}$-complet (1/2)

st-CONN est dans $\mathcal{NL}$

Un certificat pour $\langle G, s, t \rangle$ est un chemin reliant s à t dans G.
Un vérificateur pour un tel certificat parcourt le chemin en comptant le nombre de noeuds visités.

- Il retourne vrai si le chemin atteint bien t en passant par moins de n noeuds.
- Il retourne faux si le chemin s'arrête avant d'atteindre t ou s'il n'a pas atteint t en traversant n noeuds.

Pourquoi est-ce faisable en espace logarithmique ?

- La taille de l'entrée est strictement plus grande que n.
- Il faut $log(n)$ en espace pour maintenir le compteur et $2log(n)$ espace pour maintenir le noeud courant et le suivant.
- La vérification de l'existence de l'arête se fait en regardant le graphe, et donc ne consomme pas d'espace supplémentaire.

st-CONN est $\mathcal{NL}$-complet (2/2)

Tous les problèmes dans $\mathcal{NL}$ se réduisent à st-CONN

Soit $L \in \mathcal{NL}$. On construit le graphe de configuration de la machine vérifiant L pour l'entrée x. Vérifier x revient à vérifier l'existence d'un chemin entre le noeud de départ et celui d'arrêt du graphe de configuration, donc à st-CONN.

On utilise le vérificateur de la preuve précédente. La seule question est de savoir si l'on peut bien stocker le compteur et les noeuds nécessaires, maintenant que le graphe n'est plus dans l'entrée.

- Le vérificateur de L utilise $log(n)$ espace, donc chaque noeud de son graphe de configuration se décrit en espace $O(log(n))$
- L'existence d'une arête se vérifie en espace $O(log(n))$.

Le complémentaire de $\mathcal{NL}$: $co\mathcal{NL}$

Définition

Un language $L \in \{0,1\}^*$ appartient à $co\mathcal{NL} \triangleq$ le complémentaire $L^{\mathcal{C}} = \{0,1\}^* \setminus L$ appartient à $\mathcal{NL}$
Exemple : la non-existence d'un chemin entre s et t dans G appartient à $co\mathcal{NL}$.

À noter : toute classe déterministe est égale à son complémentaire. La notion n'a donc d'intérêt que pour les classes non-déterministes, c'est-à-dire pour la vérification.

non-st-CONN est $co\mathcal{NL}$-complet

Si L appartient à $co\mathcal{NL}$, alors $L^{\mathcal{C}}$ appartient à $\mathcal{NL}$. On peut donc prendre le vérificateur M de $L^{\mathcal{C}}$, et la vérification de L se réduit à la certifier la non-existence d'un chemin entre la configuration de départ et celle d'acceptation dans le graphe de configuration de M.

$\mathcal{NL} = co\mathcal{NL}$

non-st-CONN appartient à $co\mathcal{NL}$

On cherche un certificat polynomial qui prouve la non-existence d'un chemin. Si l'on connait la nombre c_n de noeuds accessibles en n étapes à partir de s, prouver que v ne fait pas partie de ces noeuds revient à donner c_n chemins, un pour chacun des noeuds.

Comment vérifier que c_n noeuds sont accessibles en n étapes ?

- Il y a 1 noeud accessible en 0 étapes : s.
- Si l'on connait c_i le nombre de noeuds accessibles en i étapes, certifier que v est accessible ou inaccessible en $i+1$ étapes se fait avec les c_i chemins en i étapes, et en vérifiant pour chacun de ses chemins s'il finit en v ou en un voisin de v.
- Du coup, on peut vérifier qu'il y a c_{i+1} noeuds accessibles en $i+1$ étapes si l'on a déjà vérifié qu'il y a c_i noeuds accessibles en i étapes.

$co\mathcal{NL}$ et $co\mathcal{NP}$

$co\mathcal{NP}$, le complémentaire de $\mathcal{NP}$

Un language $L \in \{0,1\}^*$ appartient à $co\mathcal{NP} \triangleq$ le complémentaire $L^{\mathcal{C}} = \{0,1\}^* \setminus L$ appartient à $\mathcal{NP}$

Différence entre espace et temps pour la complémentarité

$\mathcal{NL} = co\mathcal{NL}$, mais on conjecture fortement que $\mathcal{NP} \neq co\mathcal{NP}$.

Le certificat précédent donne une idée : pour prouver qu'il n'existe pas de chemin, on doit faire une exploration presque exhaustive des chemins. Cela rentre dans $\mathcal{NL}$ parce que vérifier un chemin est très peu couteux en espace, et que l'espace se réutilise.
A l'inverse, le temps n'est pas réutilisable, et donc un tel certificat n'aurait pas de sens pour $\mathcal{NP}$

Plan

1. Différences entre l'espace et le temps
   - Détails préliminaires
   - Classes élémentaires
2. Moins que l'entrée : espace logarithmique
   - Calcul en espace logarithmique
   - Vérification en espace logarithmique
3. Surpasser tous les joueurs : espace polynomial

Pourquoi s'intéresser à l'espace polynomial ?

Les raisons

- Contient $\mathcal{P}$, puisqu'un calcul en temps polynomial n'a pas le "temps" d'utiliser un espace plus que polynomial.
- Cette quantité d'espace permet de resoudre presque tous les problèmes intéressants, puisqu'elle contient $\mathcal{P}$ et $\mathcal{NP}$.
- Permet d'étudier l'impact de la réutilisabilité de l'espace par rapport au temps.
- Capture la recherche exhaustive utilisant un espace raisonnable.

La classe $\mathcal{PSPACE}$

Définition

La classe de complexité $\mathcal{PSPACE} \triangleq \bigcup_{c \in \mathbb{N}} \mathcal{DSPACE}(n^c)$

Liens avec les autres classes connues

On a $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE}$ et $\mathcal{L} \subseteq \mathcal{NL} \subsetneq \mathcal{PSPACE}$.
Il est fortement conjecturé que toutes ces inclusions (à part peut-être $\mathcal{L} \subseteq \mathcal{NL}$) sont strictes.

Comme l'espace polynomial est au moins aussi puissant que le temps polynomial, les réductions utilisées sont des réductions en temps polynomial comme pour $\mathcal{NP}$.

Le problème fondamental de $\mathcal{PSPACE}$: TQBF

True Boolean Quantified Formula

TQBF $\triangleq$ l'ensemble des formules booléennes complètement quantifiées qui sont vraies.
Exemple : $\forall x_1, \exists x_2 : x_1 \vee x_2$ appartient à TQBF.
TQBF est $\mathcal{PSPACE}$-complet.

On ne considérera que les formules de la forme $Q_1 x_1 ... Q_n x_n : \phi(x_1, ..., x_n)$ où les Q_i sont des quantificateurs booléens, et où ϕ est un prédicat sans aucun quantificateur.
On peut se le permettre car il est possible de transformer n'importe quelle formule totalement quantifiée en une formule de cette forme en temps polynomial.

TQBF appartient à $\mathcal{PSPACE}$

TQBF appartient à $\mathcal{PSPACE}$

L'algorithme évalue récursivement la formule :

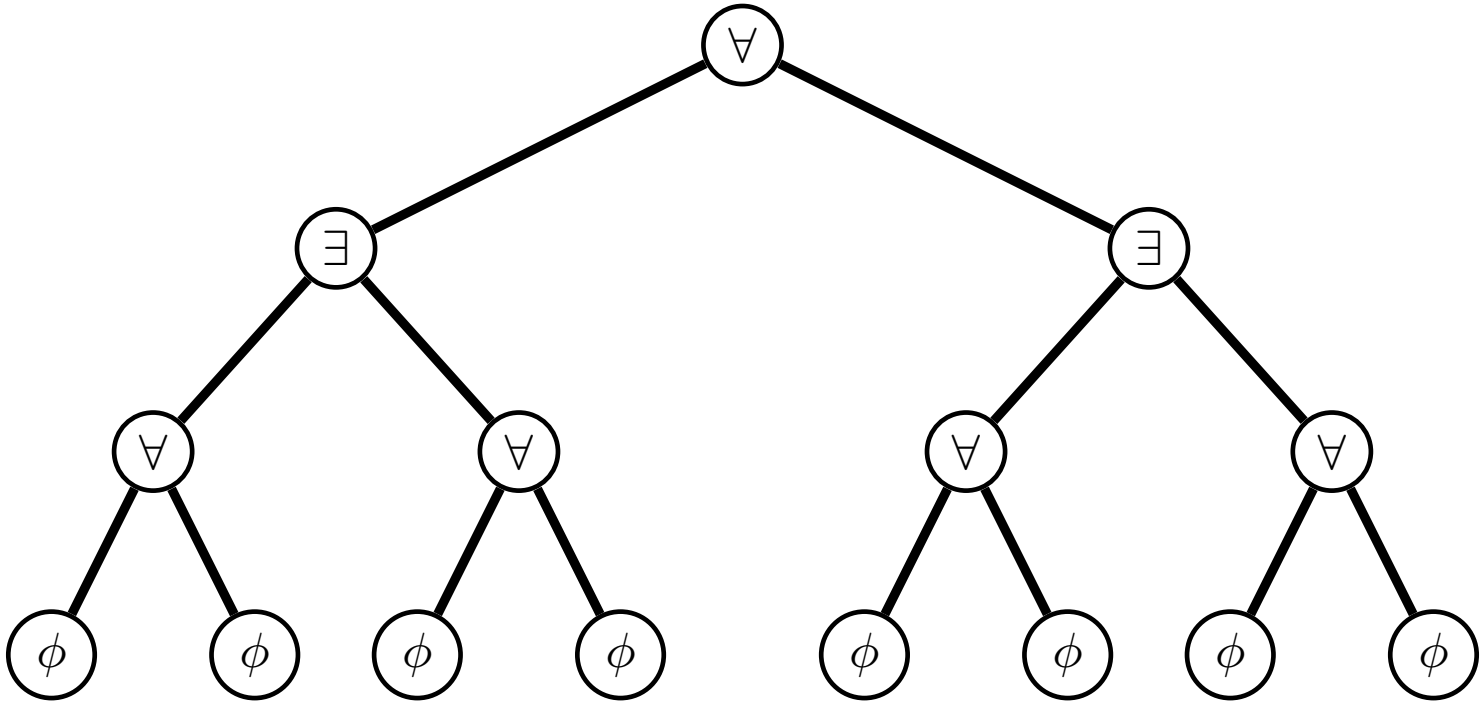
- On fixe à 0 la valeur de la première variable x quantifiée, on évalue la formule avec $x = 0$, et on note le résultat.
- On fixe x à 1, on évalue la formule avec $x = 1$ et on note le résultat
- On calcule le $\wedge$ ou le $\vee$ des deux résultats, selon que le quantificateur de x est un $\forall$ ou un $\exists$.

Soit n le nombre de variables et m la taille de la formule.

- Évaluer la formule avec toutes ses variables fixées se fait en temps polynomial en m, donc aussi en espace polynomial.
- On peut réutiliser l'espace pour les deux évaluations si l'on garde les valeurs des variables et les évaluations partielles non utilisées. Donc $O(n)$ bits, c'est-à-dire $O(m)$ bits.

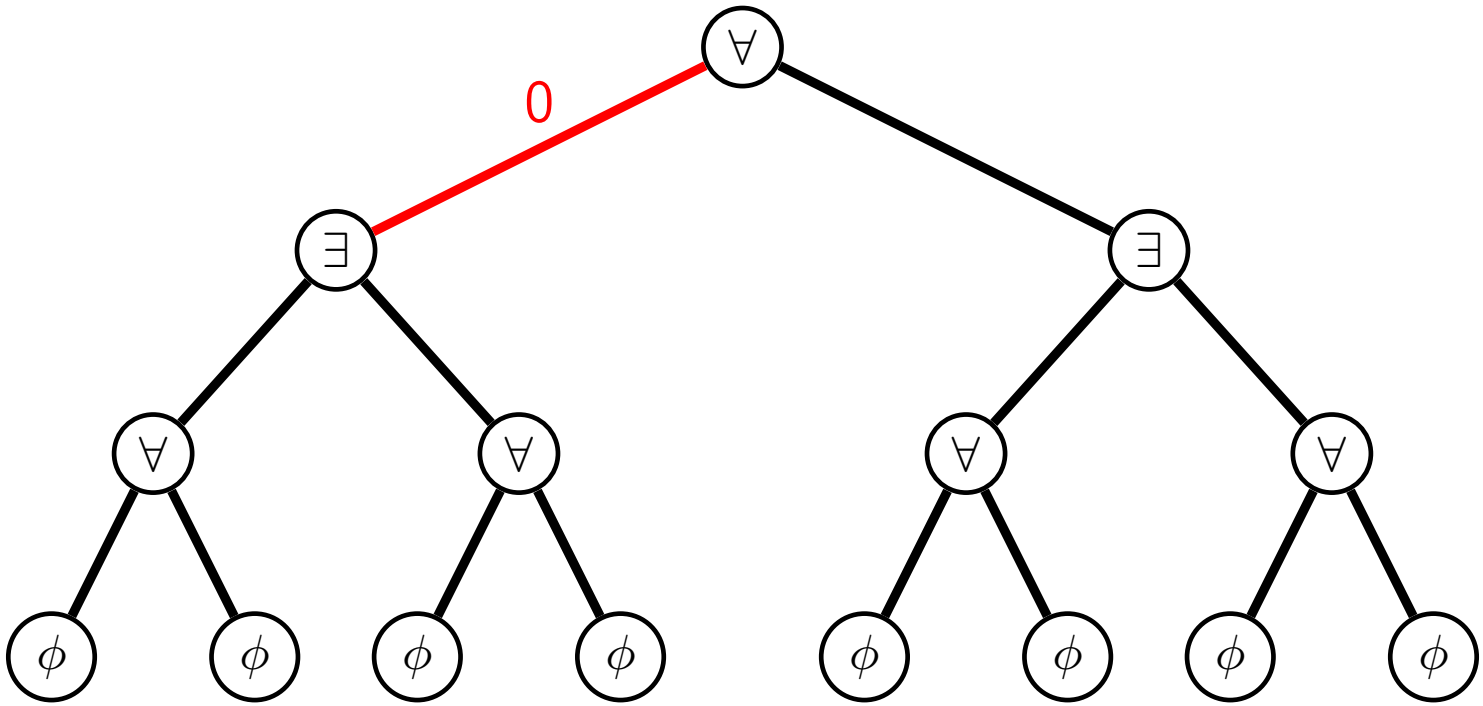
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



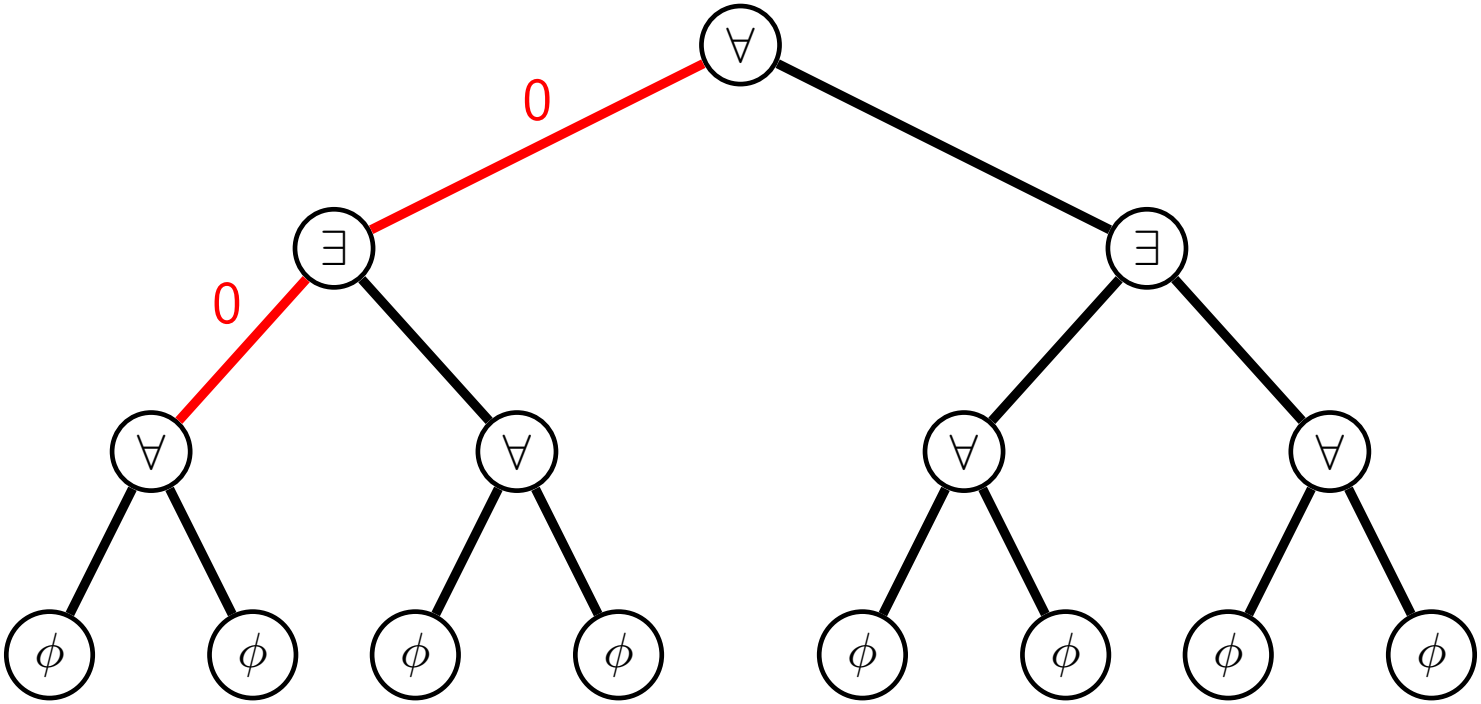
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



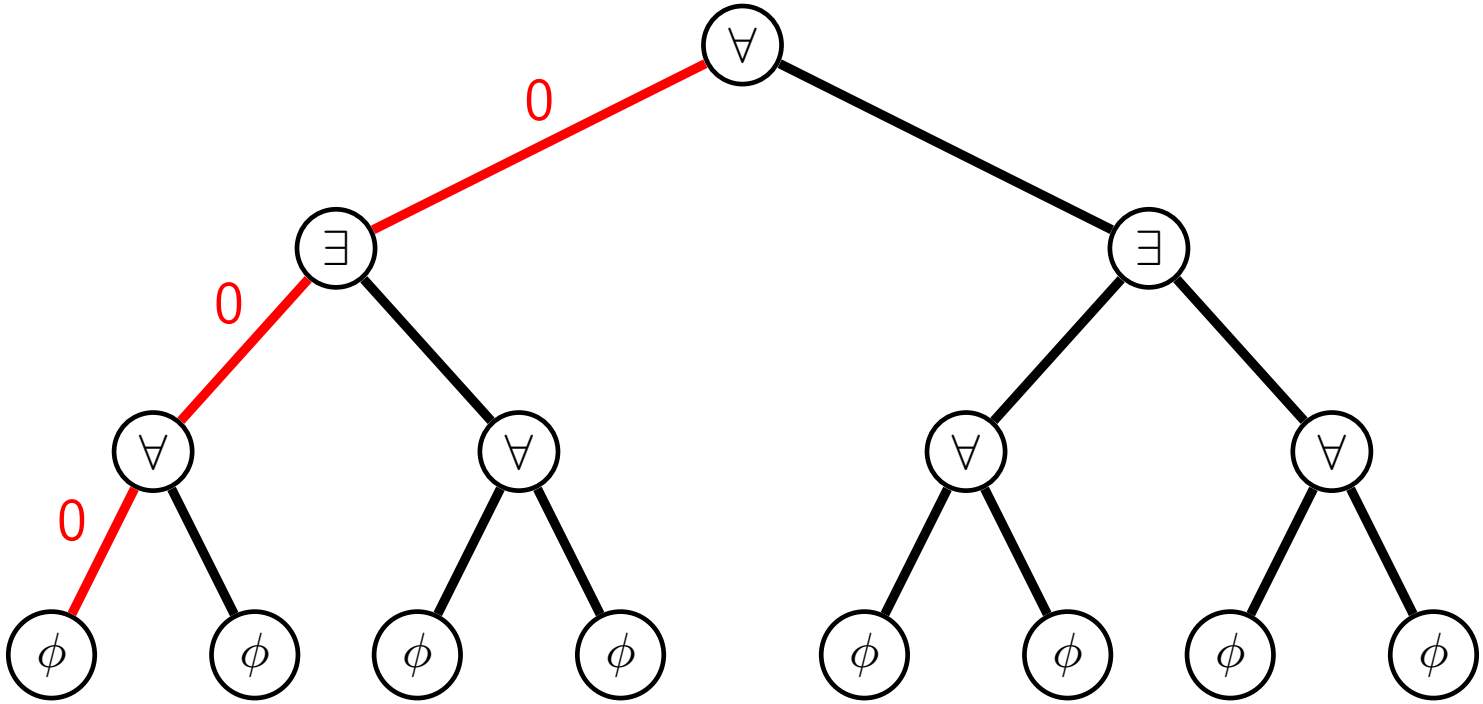
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



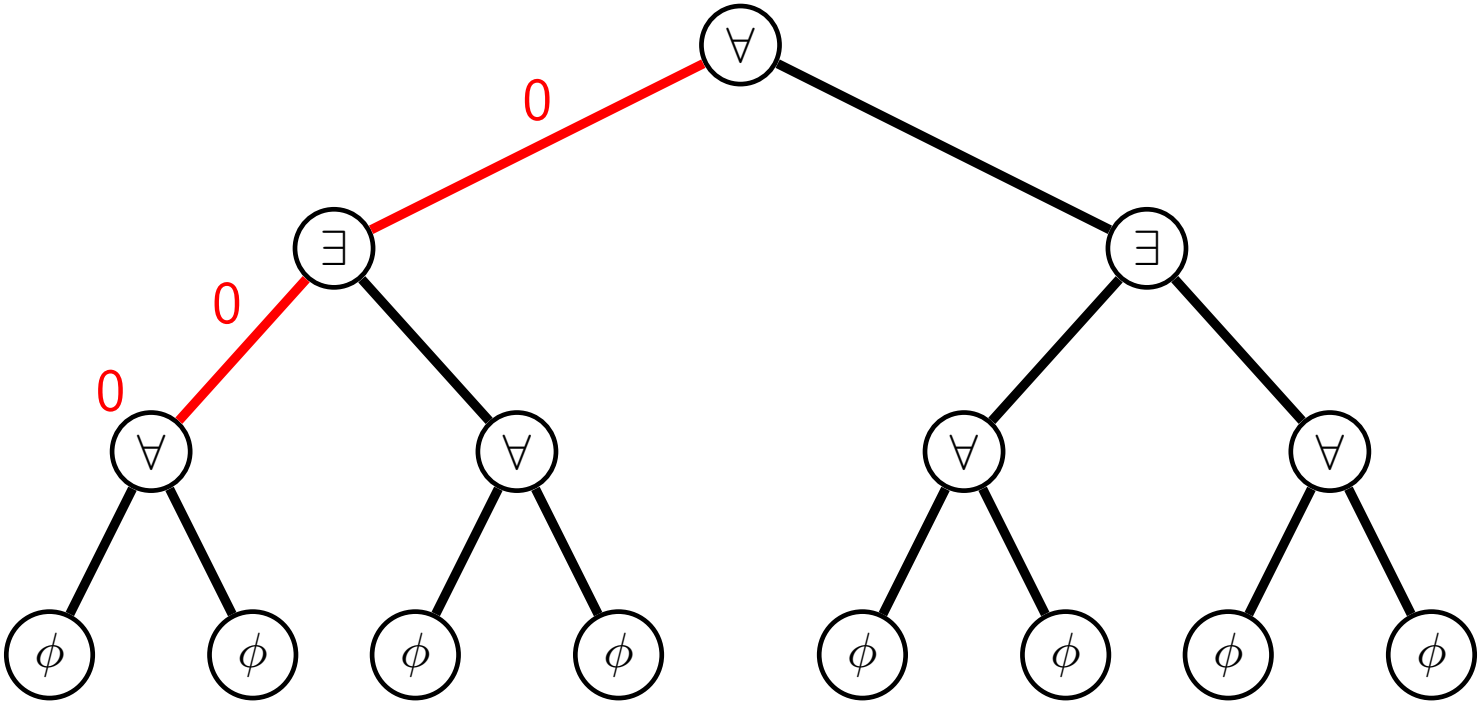
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



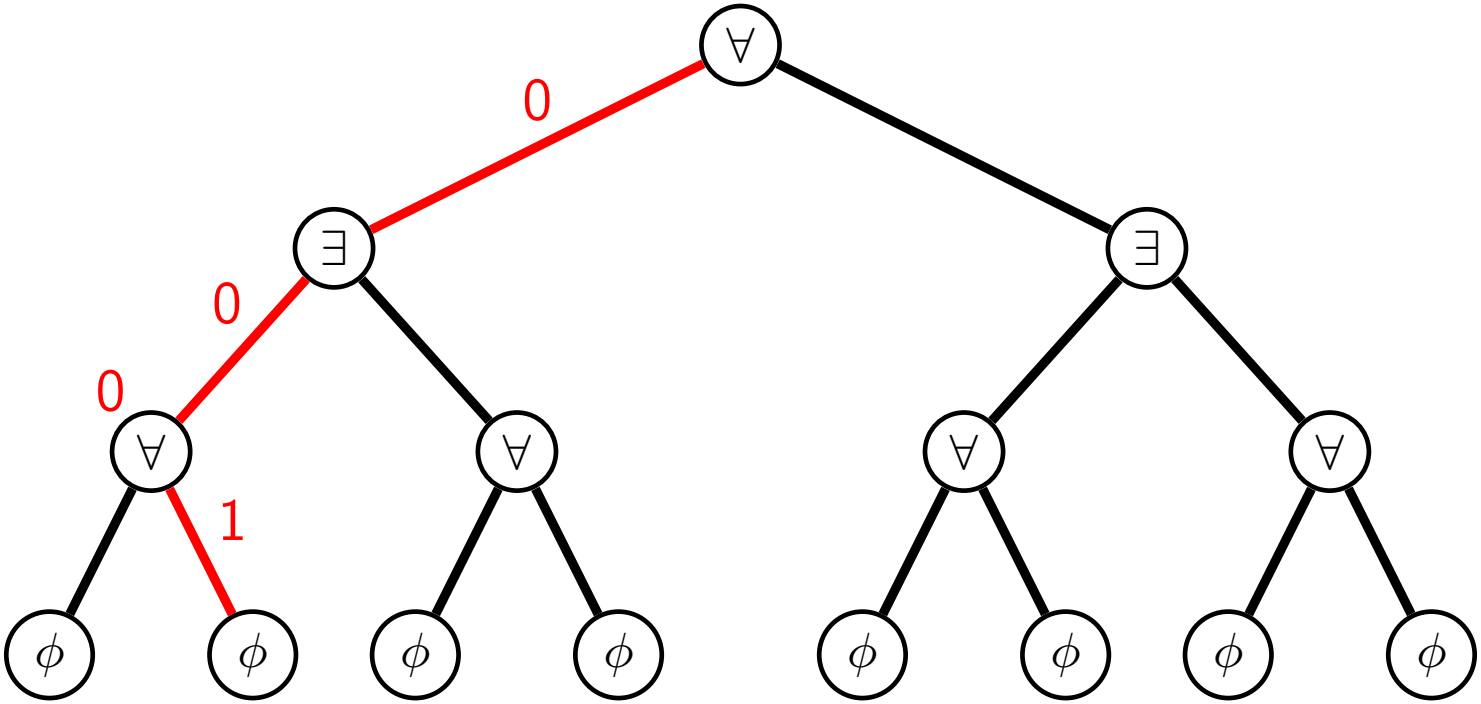
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



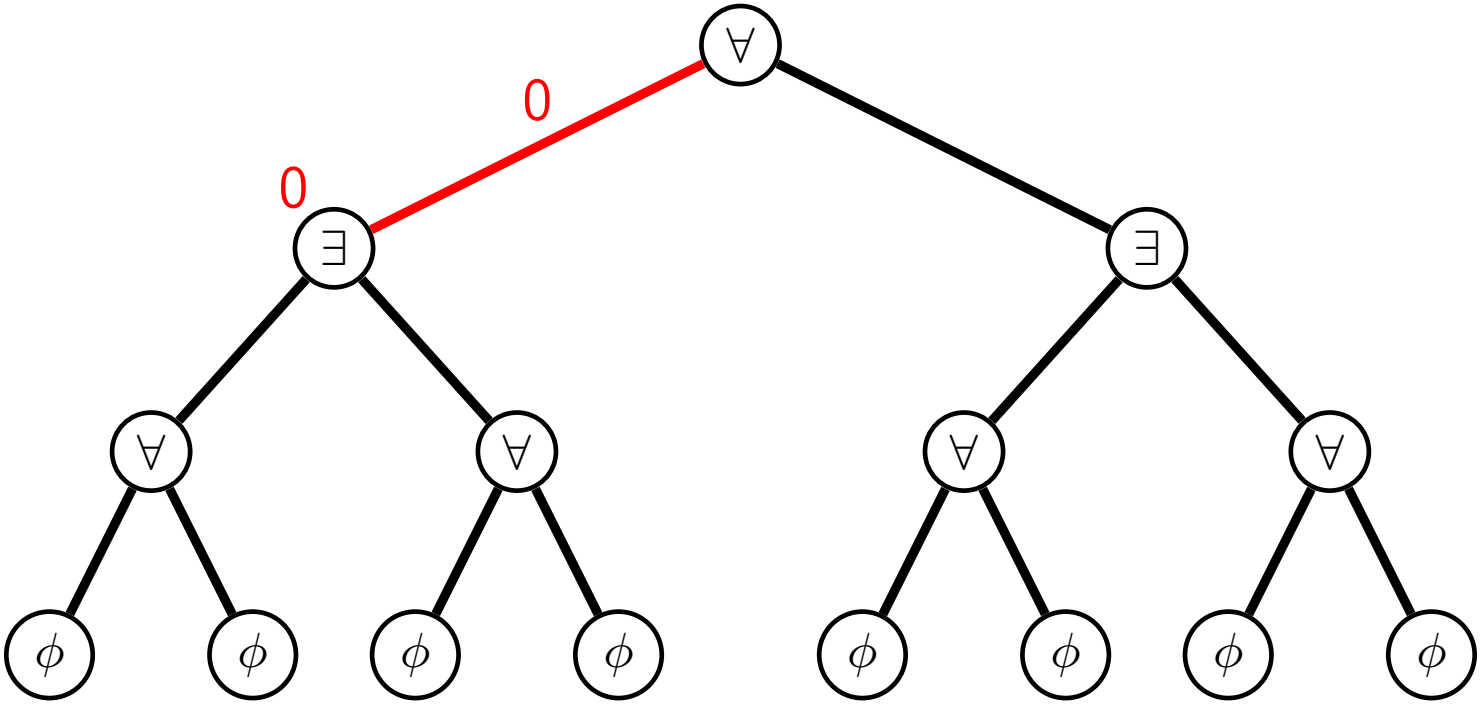
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



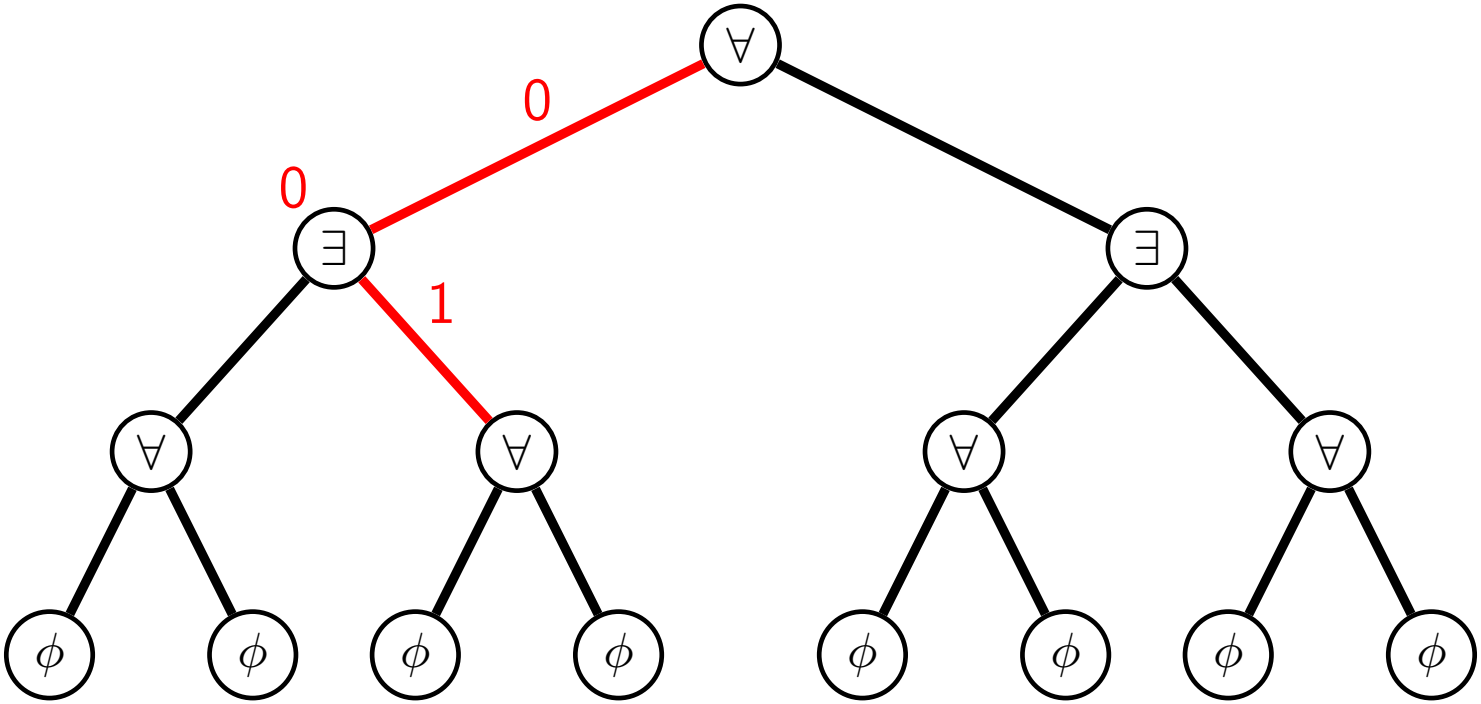
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



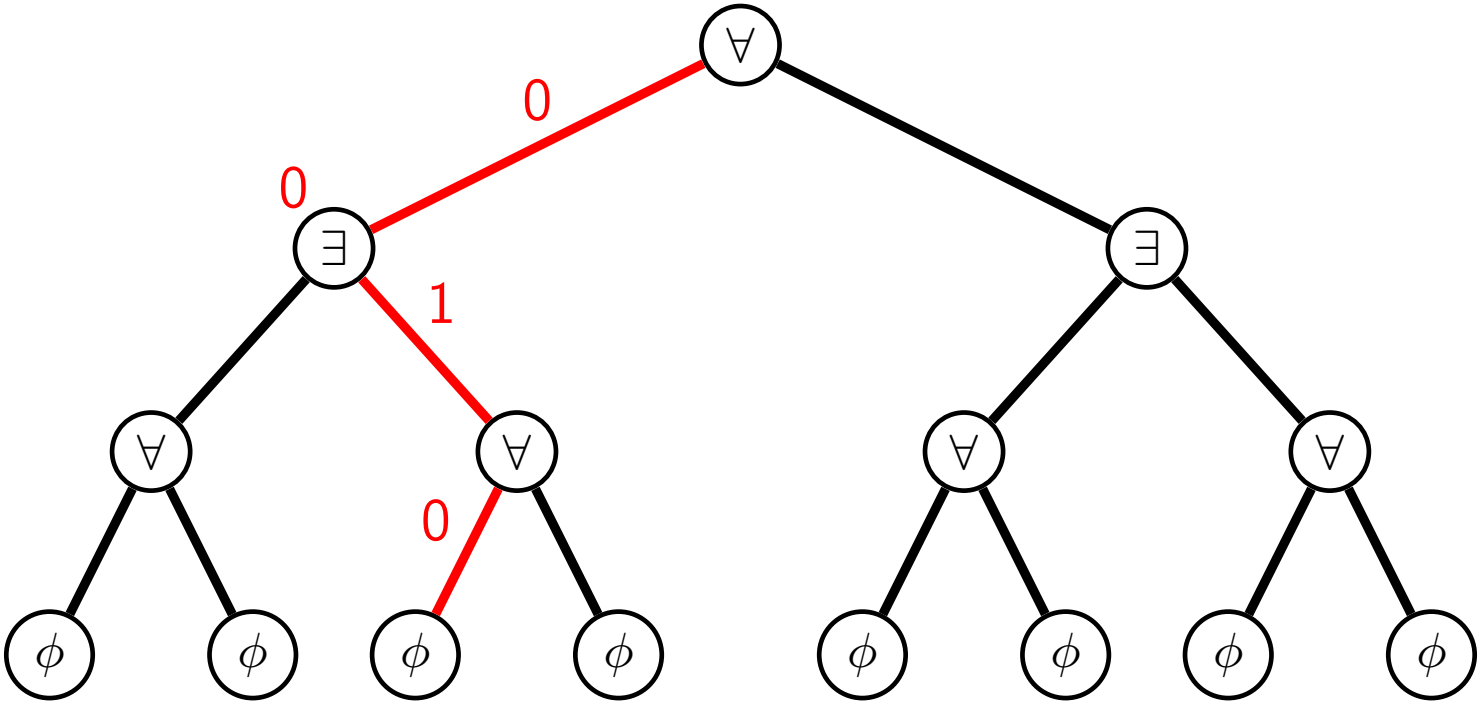
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



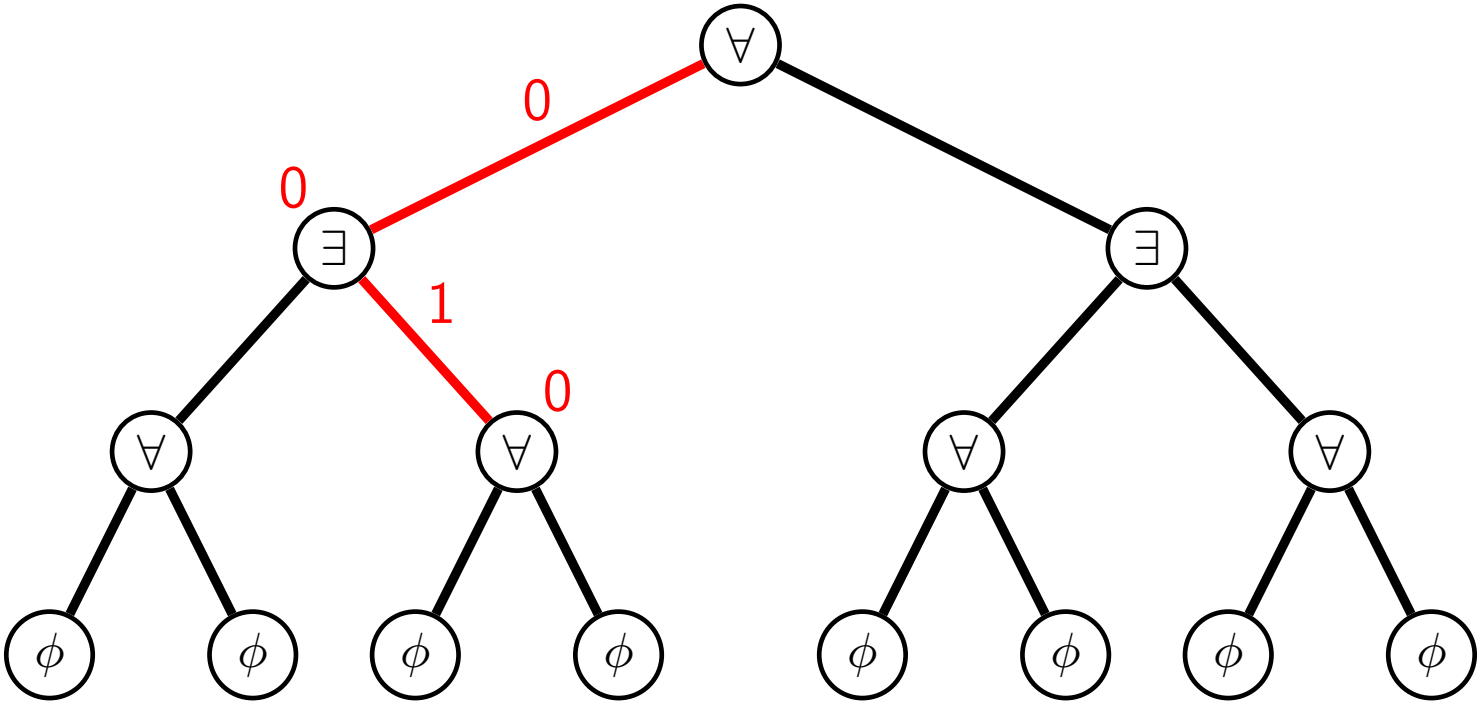
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



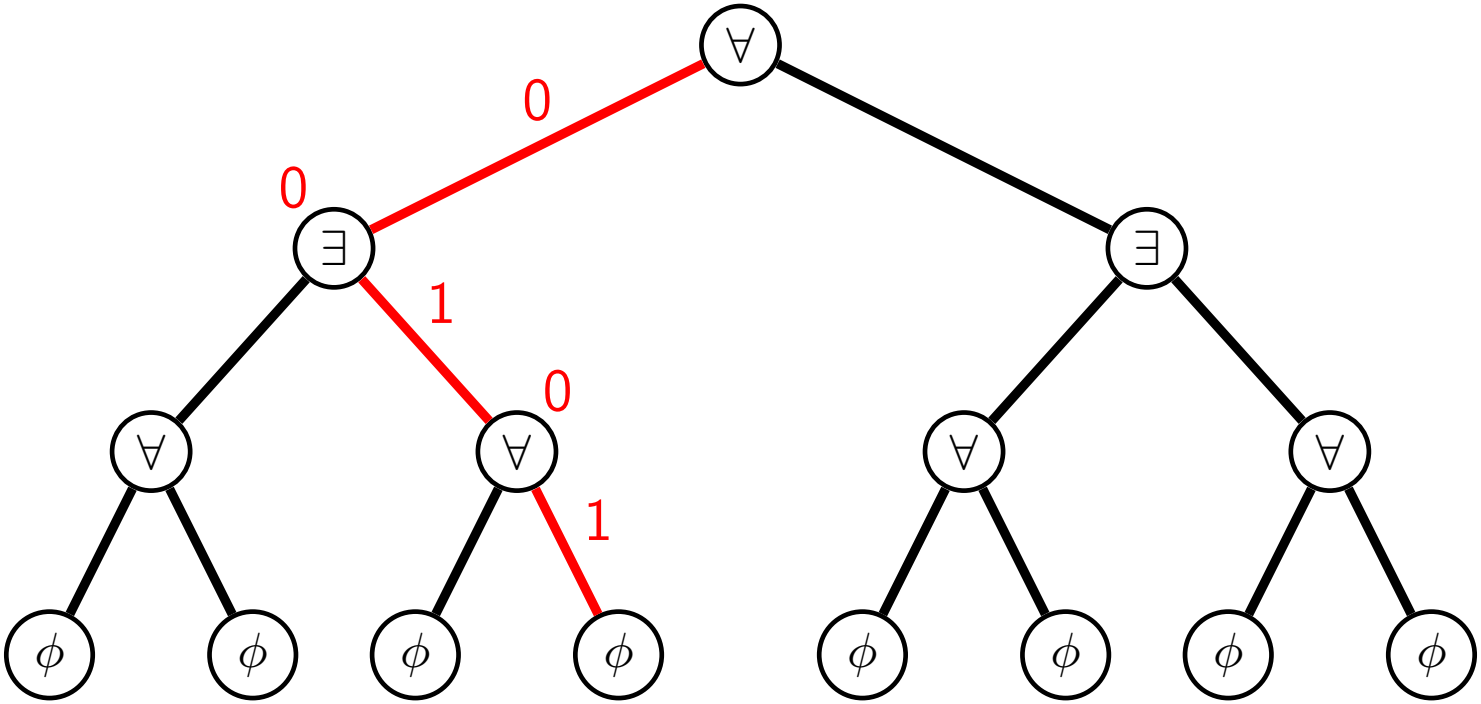
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



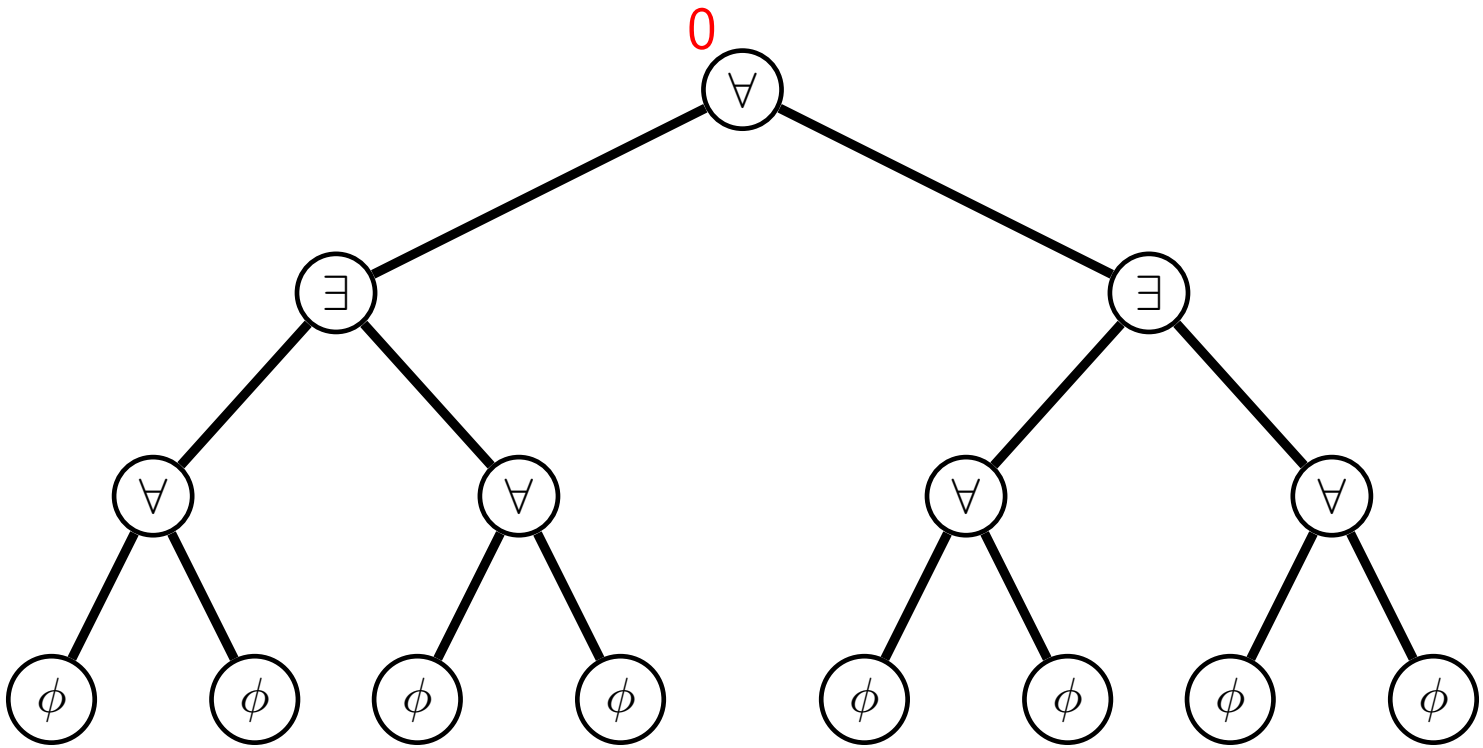
Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



Exemple simple de résolution de TQBF

$$\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$$



On peut accélérer en arrêtant l'évaluation au premier 0 pour $\forall$ et au premier 1 pour $\exists$.

TQBF est $\mathcal{PSPACE}$-difficile (1/2)

Intuition de la preuve

Soit $L \in \mathcal{PSPACE}$. Il existe donc une machine M qui décide L en espace polynomial. Soit $x \in \{0,1\}^*$.

- On construit le graphe de configuration de $M(x)$.
- Par nos résultats sur les graphes de configuration, il existe un polynome $q(n)$ tel que le graphe de $M(x)$ a $2^{q(n)}$ noeuds.
- On construit donc une formule polynomiale en n qui capture l'existence d'un chemin entre le noeud de départ et celui d'acceptation de taille $\leq 2^{q(n)}$.

TQBF est $\mathcal{PSPACE}$-difficile (2/2)

Les formules ψ_i capture l'existence d'un chemin de taille $\leq 2^i$ entre deux noeuds.

- $\psi_0(s, e)$ indique juste qu'il existe une transition entre les deux, une formule non quantifiée déjà définie dans la preuve que SAT est NP-complet, et donc polynomiale ($\leq r(n)$).
- $\psi_{i+1}(s, e)$ pourrait s'écrire $\exists m : \psi_i(s, m) \wedge \psi_i(m, e)$. Sauf que cela donnerait une formule de taille exponentielle pour $\psi_{q(n)}$. Plutôt, on écrit
  $\exists m, \forall u, v : ((u = s \wedge v = m) \vee (u = m \wedge v = e)) \implies \psi_i(u, v)$.

Pourquoi $\psi_{q(n)}$ est polynomiale

La première partie de la formule récursive utilise un nombre constant de configurations en espace polynomial, donc est en taille polynomiale ($< p(n)$).
$\psi_{q(n)}$ est donc de taille bornée par $q(n).p(n) + r(n)$.

$\mathcal{NPSPACE} = \mathcal{PSPACE}$

$\mathcal{NPSPACE} = \mathcal{PSPACE}$

La preuve de réduction des slides précédents n'utilise pas le fait que chaque noeud n'a qu'une seule arête sortante.
Elle est fonctionne donc aussi pour le graphe de configuration d'une machine non-déterministe.
On en déduit que TQBF est $\mathcal{NPSPACE}$-complet, et donc que $\mathcal{NPSPACE} = \mathcal{PSPACE}$.

Le non-déterminisme (ou les certificats) est inutile avec de l'espace polynomial.

Après tout, on peut explorer l'espace des certificats polynomiaux en espace polynomial.

$co\mathcal{NPSPACE} = \mathcal{NPSPACE} = \mathcal{PSPACE}$

$co\mathcal{NPSPACE} = \mathcal{NPSPACE} = \mathcal{PSPACE}$

La preuve de $co\mathcal{NL} = \mathcal{NL}$ se généralise parfaitement au cas polynomial :

- Un problème complet pour $co\mathcal{NPSPACE}$ est la non-existence d'un chemin dans tout graphe satisfaisant les propriétés du graphe de configuration d'une machine à espace polynomial.
- On utilise le certificat de la preuve de $co\mathcal{NL} = \mathcal{NL}$ pour vérifier la non-existence du chemin en espace polynomial.

Donc $co\mathcal{NPSPACE} = \mathcal{NPSPACE} = \mathcal{PSPACE}$.

La preuve de $co\mathcal{NL} = \mathcal{NL}$ est encore plus généralisable.
Pour $NSPACE(S(n))$ la classe des problèmes résolubles par une machine de Turing non-déterministe en espace $O(S(n))$, on a :
$S(n) \geq log(n) \implies co\mathcal{NSPACE}(S(n)) = \mathcal{NSPACE}(S(n))$.

Jeux généralisés et stratégies

Jeux généralisés

Un jeu généralisé est une version d'un jeu qui peut se jouer sur un plateau de taille arbitrairement grande.
Par exemple, on peut généraliser le go ou les échecs sur des plateaux de taille $n * n$.

Il y a souvent plusieurs généralisations pour un même jeu, selon les limites du nombre de mouvements ou les règles pour la répétition d'une position.

Stratégie gagnante

Une stratégie gagnante pour un jeu dans une position donnée est une stratégie telle que le joueur l'utilisant gagne quelque soient les mouvements de l'adversaire.

PSPACE et les stratégies gagnantes

Trouver une stratégie gagnante est $\mathcal{PSPACE}$-complet

Pour certains jeux généralisés (comme le Go) avec un nombre polynomal de mouvements et une description polynomiale des positions et des mouvements, l'ensemble des positions avec une stratégie gagnante est $\mathcal{PSPACE}$-complet.
On réduit TQBF à un jeu généralisé entre

- un joueur E qui décide la valeur des variables quantifiées par $\exists$.
- un joueur A qui décide la valeur des variables quantifiées par $\forall$.

Dans ce jeu, la valeur de la formule est l'existence ou non d'une stratégie gagnante pour le joueur existentiel.

Par exemple, savoir si $\forall x_1 \exists x_2 \forall x_3 : (x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$ appartient à TQBF revient à demander s'il existe une stratégie gagnante pour E après que A ait joué au moins un coup (x_1) de sorte que $(x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$ soit vrai.

Jeux contre la "Nature"

D'un adversaire à la "Nature"

On peut interpreter tout problème dans $\mathcal{PSPACE}$ comme un jeu contre un adversaire, qui veut donc nous faire "perdre".
Une généralisation de cette idée est que même si l'adversaire est probabiliste, alors déterminer une stratégie gagnante avec une certaine probabilité reste $\mathcal{PSPACE}$-complet.

Exemple : STQBF

STQBF est l'ensemble des formules totalement quantifiées vraies où $\forall$ est remplacé par R, un quantificateur signifiant que la variable est choisie au hasard selon une distribution donnée.
Les formules sont de la forme :
$Q_1 x_1 ... Q_n x_n : Pr[\phi(x_1, ..., x_n)] \geq 1/2$.

Conclusion

Pour résumer

L'espace se comporte fondamentalement différemment du temps, notamment par sa réutilisabilité. Cela a plusieurs conséquences :

- Un espace plus petit que la taille de l'entrée permet tout de même de résoudre des problèmes non triviaux.
- La vérification avec un espace au moins logarithmique permet de certifier les problèmes complémentaires : $S(n) \geq log(n) \implies co\mathcal{NSPACE}(S(n)) = \mathcal{NSPACE}(S(n))$.
- L'espace polynomial parait plus puissant que même la vérification en temps polynomial.
- Le non-déterminisme (ou les certificats) n'apporte rien à l'espace polynomial : $\mathcal{PSPACE} = \mathcal{NPSPACE}$.
- L'espace polynomial capture les stratégies gagnantes dans des jeux contre des adversaires ou contre la "Nature".