Le problème 2SAT

Julie Parreaux

2018-2019

Référence du développement : Papadimitriou [1, p.184 ; 398]

Leçons où on présente le développement : 915 (Classes de complexité) ; 916 (Logique propositionnelle).

1 Introduction

Le problème SAT est dans NP, il est même NP-complet (par le théorème de Cook). Une manière de calculer la satisfiabilité d'une formule du calcul propositionnelle est d'en réduire son expressivité en considérant 2SAT (on restreint les instances du problème).Ce nouveau problème portant sur les formule 2CNF est dans P. On a même mieux que cela, 2SAT est NL-complet.

Définition. On définit le problème 2SAT sur les formules propositionnelles.
Problème 2SAT
entrée : φ une formule sous forme normale conjonctive avec deux littéraux par clause
sortie : Oui si φ est satisfiable ; non sinon

Remarques sur le développement

Le développement établit que 2SAT est dans P. Si le temps (et le niveau) le permet on peut aller jusqu'à montrer que ce problème est NL-complet.

— Problème 2SAT est dans P.
  1. Écriture de l'algorithme.
  2. Preuve de sa correction.
  3. Preuve de sa complexité.

— Problème 2SAT est NL-complet.
  1. 2SAT est NL : accessibilité dans un graphe.
  2. 2SAT est NL-dur : réduction via la preuve dans P du problème d'accessibilité dans un graphe.

2 Le problème 2SAT est dans P

On commence par montrer que le problème 2SAT est dans P. Le graphe d'implication est une structure de données nous permettant de calculer une valuation pour la formule. C'est grâce à ce graphe que nous avons cette complexité pour le problème 2SAT.

Définition. Soit φ une formule sous forme 2CNF. Le graphe d'implication G_φ est le graphe où
— les sommets sont les variables propositionnelles de φ et leurs négations (les littéraux de φ) ;
— $(\alpha, \beta) \in A$ si $\alpha \to \beta$ est une clause de φ (donc $\neg\alpha \vee \beta$ est la clause que nous considérons).

Théorème. *Soit φ une formule du calcul propositionnelle. φ est insatisfiable si et seulement s'il existe une variable x telle qu'il existe un chemin de x à $\neg x$ et un chemine de $\neg x$ à x dans le graphe d'implication G_φ.*

Démonstration. $\Leftarrow$ On suppose que ces deux chemins existent pour x. Raisonnons par l'absurde : supposons que φ est satisfiable pour une valuation ν. On suppose que $\nu(x) = \text{vrai}$ (analogue dans le cas $\nu(x) = faux$).

- $\rightarrow$ ν est une valuation et $\nu(x) = \text{vrai} : \nu(\neg x) = \text{faux}$.
- $\rightarrow$ Existence d'un chemin de x à $\neg x : \exists(\alpha, \beta)$ un arc tel que $\nu(\alpha) = \text{vrai}$ et $\nu(\beta) = \text{faux}$.
- $\rightarrow$ $(\alpha, \beta) \in A : (\alpha, \beta)$ est une clause $\neg\alpha \vee \beta$.
- $\rightarrow$ $\nu(\neg\alpha \vee \beta) = \text{faux}$: contradiction.

$\Rightarrow$ On suppose qu'il n'existe pas de tels chemin pour toute variable x. Construisons une valuation pour la formule φ tel que pour tout nœuds $G(\varphi)$ il n'existe pas d'arc allant d'une variable mise à vrai à faux. L'algorithme 1 nous permet de construire cette valuation.

Algorithm 1 Algorithme permettant de construire une valuation pour une formule 2CNF via son graphe d'implication.

Entrée : G le graphe d'implication d'une formule 2CNF

```
1:  function ConstruireValuation(G)
2:      while il existe une variable non-assignée x telle que le chemin de x à ¬x n'existe pas do
3:          Choisir x vérifiant ces conditions
4:          ν(x) ← vrai
5:          for tout α ∈ Adj(x) do
6:              ν(α) ← vrai
7:              ν(¬α) ← faux
8:          end for
9:      end while
10: end function
```

(non nécessaire par hypothèse)

- $\rightarrow$ Le corps de la boucle while est bien défini car s'il existe un chemin de α à β et un chemin de α à $\neg\beta$, il existe un chemin de α à $\neg\alpha$. (Par symétrie : $(\alpha, \beta) \in A \Leftrightarrow (\neg\alpha, \neg\beta) \in A$, puis par récurrence sur la longueur du chemin.)
- $\rightarrow$ S'il existe un chemin de α à β tel que $\nu(\beta) = \text{faux}$ à une étape précédente alors $\nu(\alpha) = \text{faux}$ à cette étape. Par symétrie $(\alpha, \beta) \in A \Leftrightarrow (\neg\alpha, \neg\beta) \in A$. Si $\nu(\beta) = \text{faux}$, $\nu(\neg\beta) = \text{vrai}$ à l'étape précédente. Comme, on a traité $\neg\beta$, $\nu(\neg\alpha) = \text{vrai}$ et $\nu(\alpha) = \text{faux}$.
- $\rightarrow$ Lorsque tous les nœuds ont une valuation, il n'existe pas d'arcs tels que vrai $\rightarrow$ faux.
  - — assignation : vrai $\rightarrow$ vrai
  - — β soit mis à vrai car x

□

Proposition. *Le problème 2SAT est dans P.*

Démonstration. L'algorithme permettant de calculer la valuation d'une telle formule utilise un graphe : le graphe d'implication de la formule.

On considère l'algorithme Sat2SAT qui en temps polynomial (on est même en temps linéaire) donne la satisfiabilité d'une formule 2CNF (on résout 2SAT).

Lemme. *L'algorithme Sat2SAT appliqué à la formule φ sous forme 2CNF (Algorithme **??**) retourne satisfiable si et seulement si φ est satisfiable.*

Démonstration. $\Leftarrow$ Soit V une valuation telle que $V \models \varphi$. Par l'absurde, on suppose qu'il existe p tel que p et $\neg p$ soient dans la même composante fortement connexe. Sans perte de généralité, supposons que $V \models p$.

- $\rightarrow$ Il existe un chemine de p à $\neg p$, noté $(l_0, \ldots, l_n)$ dans G_φ (p et $\neg p$ sont dans la même composante fortement connexe).
- $\rightarrow$ $V \models l_i \rightarrow l_{i+1}$ (par définition des arcs dans le graphe des implications).

On montre par récurrence sur $i \in \mathbb{N}$ tel que $V \models l_i \forall i$. Donc $V \models \neg p$. Contradiction.

$\Rightarrow$ Supposons que l'algorithme Sat2SAT renvoie satisfiable. On considère l'algorithme ConstruireValuation (Algorithme 1). On pose $V = \text{ConstruireValuation}(G)$.

- $\rightarrow$ L'opération "V de tous les littéraux de C : vrai" est bien définie. Pour toute variable p, p et $\neg p$ ne sont pas dans la même composante fortement connexe (l'algorithme Sat2SAT renvoie satisfiable) donc on ne donne une unique valeur à p ($\neg p$ n'apparaît pas dans la clause et sa valuation n'est pas mise à vrai).

→ V est une valuation totale sur les variables propositionnelles apparaissant dans φ.

- On considère tous les sommets de G_φ
- $V \models \varphi$ On considère une clause $l_1 \to l_2$ de φ. Montrons que $V \models l_1 \to l_2$. Supposons que $V \models l_1$. Ainsi, l_1 a été mis à vrai dans ConstruireValuation : on avait $l_1 \in C_1$ où C_1 est une composante fortement connexe dans un certain graphe G_1 :
  — si l_1 et l_2 sont dans une composante fortement connexe de G_φ, alors l_2 est mis à vrai comme l_1 (en même temps puisqu'il sont dans la même composante fortement connexe).
  — sinon, dans G_φ, l_1 et l_2 sont dans deux composantes fortement connexes distinctes et l_2 est dans une composante fortement connexe qui sera final avant l_1 (puisque celle de l_1 admet une transition allant à celle de l_2, l'arc (l_1, l_2)). Donc au moment où on affecte la valeur vrai à l_1, l_2 a déjà été supprimé (car la composante fortement connexe de l_1 ne peut pas être finale tant que celle de l_2 n'a pas été supprimée).
  (Fin de la preuve à revoir : il faut également prouvé que nous avons mis l_2 à vrai lors de sa suppression (qu'il a bien été supprimé car il était dans une composante finale et non car sa négation a été affecté avec une valeur...))

□

Donc le problème 2SAT est dans P. □

3 2SAT est NL-complet

On peut même montrer mieux que cela : on peut montrer que le problème 2SAT est un problème NL-complet.

Théorème. *Le problème* 2SAT *est un problème NL-complet.*

Démonstration. **2SAT est dans** NL On simule le problème d'accessibilité dans le graphe d'implication à l'aide des clauses (on n'a pas besoin de la construire) pour vérifié si une variable et sa négation sont dans la même composante fortement connexe.

2SAT est NL**-dur** On réduit NON-ACCESSIBILITÉ à 2SAT ($NL = co - NL$ et ACCESSIBLE est NL-complet). On suppose que G est un graphe acyclique (ne change rien à se complexité).

$\forall (x, y) \in A$, on construit une clause $(\neg x \vee y)$ dans φ :

— une variable par noeud du graphe ;

— deux clauses supplémentaires : s et $\neq t$ pour le noeud source et cible.

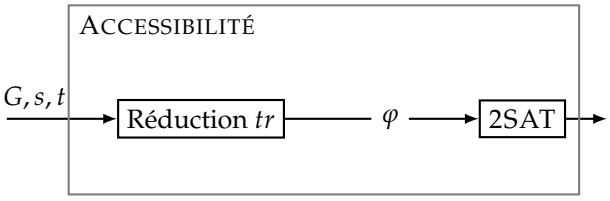

FIGURE 1 – Schéma du principe de réduction du problème NON-ACCESSIBILITÉ au problème 2SAT.

φ est satisfiable si et seulement s'il n'existe pas de chemin de s à t.

⇒ Supposons que φ est satisfiable. Raisonnons par l'absurde et supposons qu'il existe un chemin de s à t : $s, x_1, \ldots, x_n, t$. Alors $v(s) = v(x_i) = v(t)$ pour tout $i \in \{1, \ldots, n\}$. Contradiction.

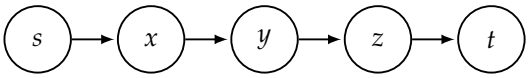

FIGURE 2 – Exemple d'un chemin de taille 3 dans un graphe.

Exemple de la formule correspondant à ce chemin : $s \wedge (\neg s \vee x) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg z \vee t) \wedge \neg t$

⇐ Raisonnons par contraposée : supposons qu'il existe un chemin de s à t. Alors comme précédemment, $v(s) = v(x_i) = v(t)$ pour tout $i \in \{1, \ldots, n\}$. On en déduit que φ est insatisfiable.

De plus cette réduction s'effectue en place constante puisque le graphe encode la formule (d'où en log space). D'où le résultat de dureté. □

4 Technique de preuve : la réduction

Pour montrer qu'un problème apparaît dans une certaine classe de complexité (et même sa dureté) ou qu'il est indécidable, nous utilisons une technique de preuve : la réduction. Pour appliquer le principe de réduction il nous faut connaître un premier problème possédant les propriétés que l'on souhaite montrer sur le deuxième.

Nous présentons ici le principe de la réduction dans sa généralité puis nous verrons comment le spécialiser pour en faire ce que nous souhaitons.

Définition. Une réduction d'un problème A à un problème B (Figure 3) est une fonction tr calculable telle que pour tout w instance de A, w est une instance positive de A si et seulement si $tr(w)$ est une instance positive de B. On note $A \leq B$.

On dit que A se réduit à B s'il existe une réduction de A à B (intuitivement, A est plus facile que B).

Remarque. En fonction des propriétés sur la fonction tr, on obtient différentes réductions qui vont nous permettre de spécialiser la réduction au résultat que nous souhaitons montrer.

Théorème (Principe de la réduction)**.** *Si A se réduit à B, alors si P est une propriété sur B alors P est une propriété sur A (dans notre cas, P peut être l'appartenance à une classe de complexité ou être le caractère indécidable d'un problème, ...).*

Démonstration. On raisonne par l'absurde et grâce à la fonction de traduction, on obtient une contradiction. □

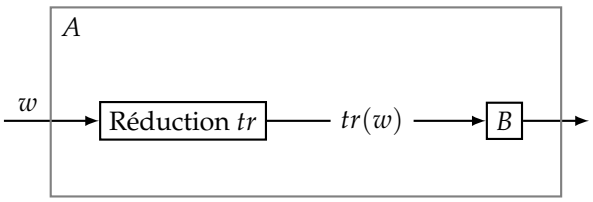

FIGURE 3 – Schéma du principe de réduction du problème A au problème B.

Nous allons donner quelques réductions de A à B et leurs propriétés. On note C une classe de complexité telle que $P \subseteq C$.

Réduction	Propriétés	Conséquences
Calculable	tr est calculable par une machine de Turing (variante de MT : équivalence)	A indécidable $\Rightarrow$ B indécidable B décidable $\Rightarrow$ A décidable
Temps polynomial	tr est calculable par une machine de Turing déterministe en temps polynomial	A est C-dur $\Rightarrow$ B est C-dur $(C \neq P)$ $B \in C \Rightarrow A \in C$
Espace logarithmique	tr est calculable par une machine de Turing déterministe en espace logarithmique (la MT a trois rubans)	A est D-dur $\Rightarrow$ B est D-dur $B \in D \Rightarrow A \in D$ $(D \neq P)$ avec $D \in \{L, NL, co-NL, P\}$

Remarque. La réduction en espace logarithmique est une réduction très spéciale car une machine qui travail en espace logarithmique a au moins deux rubans (il ne faut pas que l'entrée rentre dans la calcul). Mais pour la réduction, la sortie n'est pas non plus dans la borne de la mémoire utilisé (notons qu'elle est polynomiale (car une réduction en espace logarithmique s'exécute en temps polynomial)), il nous faut donc un troisième ruban. La machine de Turing effectuant la réduction a donc trois rubans : un ruban contenant l'entrée en lecture seule et en une seule passe ; un ruban de travail logarithmique et un ruban de sortie polynomial en écriture seule et en une seule passe.

Proposition. *Une réduction en espace logarithmique est une réduction en temps polynomial.*

Références

[1] C.H. Papadimitriou. *Computational complexity*. Pearson, 1993.