Méthode des tableaux : Optimisation de la determination de la satisfaisabilité en logique propositionelle

Contents

1 Introduction

1.1 Definition

Definition 1: La *logique propositionelle* est un type de logique où les formules sont obtenus par des variables propositionelles reliées par des connecteurs.

Definition 2 (Modèle): Un *modèle* d'une formule ϕ est une valuation qui rend vraie cette formule. On note l'ensemble des modèles de ϕ par:

$$Mod(\phi) := \{v \in Val | v \vDash \phi\}$$

Val étant l'ensemble des valuations de ϕ et $v \vDash \phi$ signfiant que la valuation v satisfait ϕ

Definition 3 (Conséquence Logique): Une formule ϕ est *conséquence logique* d'une formule, notée ψ si $Mod(\psi) \subseteq Mod(\phi)$. On note cela $\psi \vDash \phi$

Definition 4 (Arbre de déduction): Arbre dont les sommets sont composés de formule, qui sont soit une hypothèse à la racine de l'arbre, soit une formule obtenue par l'application d'une règle sur une formule présente dans la même branche plus proche de la racine.

Definition 5 (Branche fermée): Une branche est fermée si elle contient ϕ et $\neg\phi$

Definition 6 (Arbre fermé): Un arbre de déduction est fermé si toutes les branches le sont.

1.2 Principe

On note $n \in \mathbb{N}^*$, le procédé de la méthode des tableaux consiste donc à séparer les formules logiques complexes en plus petite formule jusqu'à que des pairs complementaires de litteraux (a et $\neg a$) soit extrait ou qu'on ne peut plus simplifier la formule. On en déduit ainsi si une formule ϕ est insatisfaisable ou pas en adoptant des règles qu'on applique sur un arbre de déduction.

- On place $\neg\phi$ dans la racine de l'arbre.
- On applique les règles (R_x) suivantes à chaque branche non fermé de l'arbre
- Si l'arbre est in fine fermé, alors ϕ est vrai

Les règles (R_x) dites Smullyan-style sont:

$$\begin{array}{c} \phi \vee \psi \\ \swarrow \quad \searrow \\ \phi \qquad\quad \psi \end{array} \ (R_{\vee}) \qquad \begin{array}{c} \neg(\phi \wedge \psi) \\ \swarrow \quad \searrow \\ \neg\phi \qquad\quad \neg\psi \end{array} \ (R_{\neg\wedge}) \qquad \begin{array}{c} \phi \implies \psi \\ \swarrow \quad \searrow \\ \neg\phi \qquad\quad \psi \end{array} \ (R_{\implies}) \qquad \begin{array}{c} \neg\neg\phi \\ | \\ \phi \end{array} \ (R_{\neg\neg}) \qquad \begin{array}{c} \phi \wedge \psi \\ | \\ \phi \\ \psi \end{array} \ (R_{\wedge})$$

$$\begin{array}{c} \neg(\phi \vee \psi) \\ | \\ \neg\phi \\ \neg\psi \end{array} \ (R_{\neg\vee}) \qquad \begin{array}{c} \neg(\phi \implies \psi) \\ | \\ \phi \\ \neg\psi \end{array} \ (R_{\neg\implies}) \qquad \begin{array}{c} \phi \Leftrightarrow \psi \\ \swarrow \quad \searrow \\ \phi \qquad \neg\phi \\ \psi \qquad \neg\psi \end{array} \ (R_{\Leftrightarrow}) \qquad \begin{array}{c} \neg(\phi \Leftrightarrow \psi) \\ \swarrow \quad \searrow \\ \phi \qquad \neg\phi \\ \neg\psi \qquad \psi \end{array} \ (R_{\neg\Leftrightarrow})$$

Dans les cas où on trouve dans une même branche a et $\neg a$, on ferme la branche, si l'arbre devient fermé, alors $\neg\phi$ est forcement faux, donc ϕ est forcement vrai.

Proposition 1: Soit Ψ un ensemble d'hypothèse et ϕ une formule, on note $\Psi \vdash \phi$ l'existence d'un arbre fermé pour $\Psi \cup \{\neg\phi\}$.

$$\Psi \vdash \phi \Leftrightarrow \Psi \vDash \phi$$

Autrement dit, appliqué la méthode des tableaux à la négation de la formule et ses hypothèses est équivalent à prouver cette formule (*Prouver dans le sens que c'est une tautologie en suppossant* Ψ)

Preuve 1: La preuve de cette proposition permet de prouver la correction de l'implémentation de base. Elle est trouvable en ligne

Cette proposition assure un moyen de prouver une formule.

Proposition 2: Soit Ψ un ensemble d'hypothèse et ϕ une formule, on note $\Psi \vdash \phi$ la non existence d'un arbre fermé pour Ψ.

$$\Psi \vdash \phi \Leftrightarrow \Psi \vDash \phi$$

Autrement dit, appliqué la méthode des tableaux est équivalent à prouver la satisfaisabilité de cette formule

Preuve 2: En ligne

Cette proposition assure un moyen de prouver la satisfaisabilité d'une formule.
On écrit donc l'algorithme de base de determination de la satisfaisabilité d'une formule

```
type formula =
    | Atom of string
    | Not of formula
    | And of formula * formula
    | Or of formula * formula

let rec expand formula =
    match formula with
    | Not (Not f) -> [[f]]
    | Not (And (f1, f2)) -> [[Not f1]; [Not f2]]
    | Not (Or (f1, f2)) -> [[Not f1; Not f2]]
    | And (f1, f2) -> [[f1; f2]]
    | Or (f1, f2) -> [[f1]; [f2]]
    | _ -> [];;

let rec has_cycle branch =
    List.exists (fun f -> List.mem (Not f) branch) branch;;

let rec tableau branches =
    match branches with
    | [] -> false
    | branch :: rest ->

    if has_cycle branch then
        tableau rest
    else
        match branch with
        | [] -> true
        | f :: fs ->

        let expansions = expand f in match expansions with
        | [] -> tableau (fs :: rest)
        | new_branches ->

        let expanded_branches = List.map (fun b -> b @ fs) new_branches in
        tableau (expanded_branches @ rest);;

let is_satisfiable formula =
    let initial_branch = [formula] in tableau [initial_branch];;
```

2 Formules de forme alternée

2.1 Definition

Definition 7: Soit $n \in \mathbb{N}^*$ et $(\alpha_k)_{k \in [|1,n|]}$ des litteraux, une formule ϕ est de forme alternée ssi

$$\phi := \alpha_1 \wedge (\alpha_2 \vee (\alpha_3 \wedge (\dots(\alpha_n))))$$

On gardera les parenthèses dans la suite pour garder le coté intuitif de cet ecriture et surtout ne pas laisser l'ambiguité sur la prioritée entre les opérateurs logiques

On remarque une CNS pour que ce type de formule soit vrai (pas satisfaisable!):

Proposition 3 (CNS de Vérité): ϕ de forme alternée est vrai ssi:

$$(\exists k \in \mathbb{N}^* \text{ minimal }, \alpha_{2k-2} \text{ OU n impair } \implies \alpha_n \text{ avec } k = \frac{n-1}{2}) \text{ ET } \forall i \in [|1,k|], \alpha_{2i-1}$$

Preuve 3: ($\Longleftarrow$) Immédiat

($\implies$) Supposons ϕ de forme alternée vrai: α_1 est forcement vrai, deux possibilités : soit α_2 est vrai, soit $\alpha_3 \wedge (\dots)$ est vrai. Dans le deuxième cas, on repète le raisonnement sur $\alpha_3 \wedge (\dots)$ qui est bien de forme alternée.

Deux cas de figure:

- On arrête donc le processus dès que un litteral indéxée par un nombre pair est vrai. Dans ce cas là, tout les indéxées impair précedents sont aussi vrai.
- Si aucun litteral pair est vrai, alors n est impair sans quoi ϕ n'est pas vrai car α_n doit être vrai. On en déduit que tout les litteraux impairs sont vraies, donc la formule est vrai.

Mais ceci n'est pas important, cela aide juste à se donner une idée de la structure de la formule, pour trouver des CNS plus intéressantes.

2.2 Satisfaisabilité

On rappelle que pour prouver la satisfaisabilité d'une formule, on montre qu'il n'existe pas d'arbre fermé. Revenons à la méthode des tableaux, en appliquant les règles à la famille de formule de forme alternée, on observe un patterne sur l'arbre résultant $\forall k \in \mathbb{N}^*$:

$$\begin{array}{c} a_{2k-1} \wedge (a_{2k} \vee A) \\ | \\ a_{2k-1} \end{array}$$

$$\begin{array}{ccc} & a_{2k} \vee A & \\ \swarrow & & \searrow \\ a_{2k} & & \mathit{Suite} \end{array}$$

où A est la suite de la forme alternée, qui est aussi de forme alterné ou un littéral.

> **Proposition 4:** Chaque arbre induit par la méthode des tableaux est un arbre binaire, tel qu'on nomme dans les nodes les hypothèses à l'étape associé de l'algorithme

> **Preuve 4:** Soit A_1 et A_2 deux arbres binaires représentant l'arbre induit par la méthode des tableaux T_1 et T_2 respectivement, etiquetés par des litteraux, on suppose qu'ils sont égaux. On peut les reconstruire par une succession d'opération sur une formule par la méthode des tableaux. Etant donné que $A_1 = A_2$, on applique les mêmes opérations, sans quoi on aura pas les mêmes arbres binaires, donc on a la même formule et les mêmes hypothèses, donc $T_1 = T_2$

L'arbre induit ; pour $n = 5$; et pour $n = 4$

$$\begin{array}{ccc} & \alpha_{2k-1} & \\ \swarrow & & \searrow \\ \alpha_{2k} & & \mathit{Suite} \end{array} \qquad \begin{array}{ccccc} & \alpha_1 & & & \\ \swarrow & & \searrow & & \\ \alpha_2 & & \alpha_3 & & \\ & \swarrow & & \searrow & \\ & \alpha_4 & & \alpha_5 & \end{array} \qquad \begin{array}{cccc} & \alpha_1 & & \\ \swarrow & & \searrow & \\ \alpha_2 & & \alpha_3 & \\ & \swarrow & & \\ & \alpha_4 & & \end{array}$$

Faisons des raisonnements pour des formules de taille $n \in [|1, 3|]$:

1. Toujours satisfaisable
2. Satisfaisable ssi $\alpha_2 \neq \neg\alpha_1$
3. Satisfaisable ssi $\neg(\alpha_1 = \neg\alpha_2 = \neg\alpha_3)$

C'est à partir du cas $n = 4$ que la satisfaisabilité devient difficile à retablir. On peut l'établir par 2-SAT, et trouver qu'il y a 4096 formules possibles avec 168 d'entres elles qui sont insatisfaisable, on sent que trouver une CNS peut être compliqué, surtout quand on va prendre des n plus hauts. De plus, ca reviendrait à résoudre une sous instance du problème k-SAT, on sent que trouver une CNS est tout bonnement impossible. Attaquons nous donc à des approches plus réaliste pour le moment.

2.3 Etude de compléxité

Etudions les points faibles de notre algorithme en terme de complexité pour pouvoir par la suite étudier des solutions à implémenter. En prenant n le nombre de branches

Premièrement, *expand* et *has_cycle* sont respectivement en $\mathcal{O}(1)$ et $\mathcal{O}(n^2)$. Ensuite, dans le cas où on tombe à chaque fois sur un cycle en analysant une branche, on a une equation de récurrence de type :

$$C(n) = n^2 \times C(n-1)$$

Soit par recurrence immédiate une méthode des tableaux en $\mathcal{O}(n!^2)$

On peut montrer que l'algorithme a une compléxité qui est très largement pire que cela, intéressons nous au point critique en terme de compléxité. Premièrement, *has_cycle*, qui est appelé à chaque appel recursif, mais aussi et surtout l'expansion.

Rappellons premièrement le code de has cycle:

```ocaml
let has_cycle branch =
    List.exists (fun f -> List.mem (Not f) branch) branch;;
```

Il regarde dans la liste représentant la branche et les hypothèses extraites si pour chaque element, il existe sa négation. Une optimisation triviale serait d'éviter les doubles tests, mais cela reste en $\mathcal{O}(n^2)$.

2.4 Preuve sous hypothèse

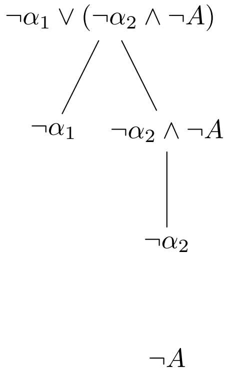
Proposition 5: Avec les mêmes définitions, soit $\phi = \alpha_1 \wedge (\alpha_2 \vee (\alpha_3 \wedge (\dots(\alpha_n))))$ une formule de formule alternée:

$$\neg\phi = \neg\alpha_1 \vee (\neg\alpha_2 \wedge (\neg\alpha_3 \vee (\dots(\neg\alpha_n))))$$

Cette forme est cohérente avec les propriétés de $\neg$.

Preuve 5: On prouve par recurrence forte sur $n \in \mathbb{N}^*$
$\mathcal{P}(n) : \neg\phi = \neg\alpha_1 \vee (\neg\alpha_2 \wedge (\neg\alpha_3 \vee (\dots(\neg\alpha_n))))$
$\mathcal{P}(1) : \neg\phi = \neg\alpha_1$ vrai
$\forall k \in \mathbb{N}^*$, supposons $\mathcal{P}(k)$:
$\neg\phi = \neg(\alpha_1 \wedge (\alpha_2 \vee (\alpha_3 \wedge (\dots(\alpha_{n+1})))))$
$= \neg\alpha_1 \vee (\neg\alpha_2 \wedge \neg(\alpha_3 \wedge (\dots(\alpha_{n+1}))))$
On trouve le résultat en appliquant l'hypothèse de recurrence sur $\alpha_3 \wedge (\dots(\alpha_{n+1}))$ qui est de forme alternée.
$\forall n \in \mathbb{N}^*, \mathcal{P}(n)$

De là, on peut enfin en déduire la forme de l'arbre induit par la méthode des tableaux sur la négation d'une formule de forme alternée sans hypothèse



Proposition 6: $\neg\phi$ de forme alternée est vrai ssi:

$$(\forall k \in \mathbb{N}^*, \neg\alpha_{2k-2} \text{ ET n impair ET } \neg\alpha_n \text{ avec } k = \frac{n-1}{2}) \text{ OU } \exists i \in [|1, k], \neg\alpha_{2i-1}$$

Preuve 6: On utilise la négation de la proposition 3 ainsi que $\phi \to F \Leftrightarrow \neg\phi \to T$

3 Logique du 1er ordre

La logique propositionelle étant mathématiquement limité, on se propose l'utilisation de la logique du 1er ordre. Cela nous permettra ainsi d'étudier Zenon, un prouveur d'automatique de theorème.

Definition 8: La *logique du 1er ordre* est un type de logique qui en plus des élements de la logique propositionelle permet l'utilisation de quantificateurs et de *termes*.

Definition 9: Soit un ensemble infini de variables $X = \{x, y, x_1, x_2, \dots\}$ et un ensemble $\mathcal{F} = \{c, f, g, \dots\}$ de symboles de fonction (autrement appelé signature). On rappelle que l'arité d'un symbole est son nombre d'argument. Les termes sont définis par induction:

- $\forall x \in X, x$ est un terme
- Tout symbole d'arité 0 (les constantes) est un terme
- $f(t_1, \dots, t_n)$ est un terme ssi f est un symbole d'arité n et $t_1, \dots, t_n$ sont des termes

On note $\mathcal{T}(\mathcal{F}, X)$ l'ensemble des termes sur $\mathcal{F}$ et X.