

Méthode des tableaux: Application en logique propositionnelle et en logique linéaire temporelle

TIPE 25/26 - Cycles et Boucles

GIL Dorian - Candidat n°20220

Intro - Positionnement

- On souhaite étudier une méthode algorithmique permettant de montrer la satisfiabilité d'une formule: la Méthode des tableaux.

$$\begin{array}{ccccc} \phi \vee \psi & \neg(\phi \wedge \psi) & \neg\neg\phi & \phi \wedge \psi & \neg(\phi \vee \psi) \\ / \quad \backslash & / \quad \backslash & | & | & | \\ \phi & \psi & \neg\phi & \neg\psi & \phi, \psi \\ & & & & \neg\phi, \neg\psi \end{array}$$

Intro - Positionnement

- On souhaite étudier une méthode algorithmique permettant de montrer la satisfiabilité d'une formule: la Méthode des tableaux.
- Elle consiste à construire un arbre avec la formule à la racine, et à utiliser des règles pour développer et créer des branches.

$$\begin{array}{ccccc} \phi \vee \psi & \neg(\phi \wedge \psi) & \neg\neg\phi & \phi \wedge \psi & \neg(\phi \vee \psi) \\ / \quad \backslash & / \quad \backslash & | & | & | \\ \phi & \psi & \neg\phi & \neg\psi & \neg\phi, \neg\psi \end{array}$$

Intro - Positionnement

- On souhaite étudier une méthode algorithmique permettant de montrer la satisfiabilité d'une formule: la Méthode des tableaux.
- Elle consiste à construire un arbre avec la formule à la racine, et à utiliser des règles pour développer et créer des branches.
- On regarde ensuite si il y a des contradictions dans toutes les branches, si c'est le cas, la formule est insatisfaisable.

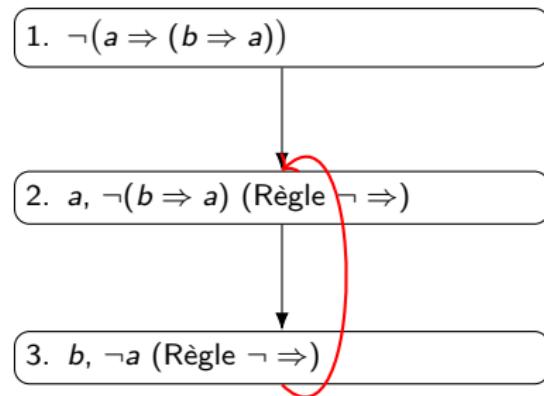
$$\begin{array}{ccccc} \phi \vee \psi & \neg(\phi \wedge \psi) & \neg\neg\phi & \phi \wedge \psi & \neg(\phi \vee \psi) \\ / \quad \backslash & / \quad \backslash & | & | & | \\ \phi & \psi & \neg\phi & \neg\psi & \neg\phi, \neg\psi \end{array}$$

Intro - Positionnement

- On souhaite étudier une méthode algorithmique permettant de montrer la satisfiabilité d'une formule: la Méthode des tableaux.
- Elle consiste à construire un arbre avec la formule à la racine, et à utiliser des règles pour développer et créer des branches.
- On regarde ensuite si il y a des contradictions dans toutes les branches, si c'est le cas, la formule est insatisfaisable.
- Cette méthode est utilisé dans diverses logiques, pour l'instant, on se restreint à la logique propositionnelle.

$$\begin{array}{cccccc} \phi \vee \psi & \neg(\phi \wedge \psi) & \neg\neg\phi & \phi \wedge \psi & \neg(\phi \vee \psi) \\ / \quad \backslash & / \quad \backslash & | & | & | \\ \phi & \psi & \neg\phi & \neg\psi & \phi & \psi & \neg\phi, \neg\psi \end{array}$$

Intro - Exemple



Contradiction: a (2) et $\neg a$ (3). Conclusion, cette formule n'est pas satisfiable.

LP - Définition de l'étude

Après l'avoir implémenter, j'ai décidé de me restreindre à une forme particulière de formule logique.

Definition (Forme Alternée)

Soit $n \in \mathbb{N}^*$, et $(a_k)_{k \in [|1, n|]}$ des littéraux, on dit que φ est de forme alternée ssi

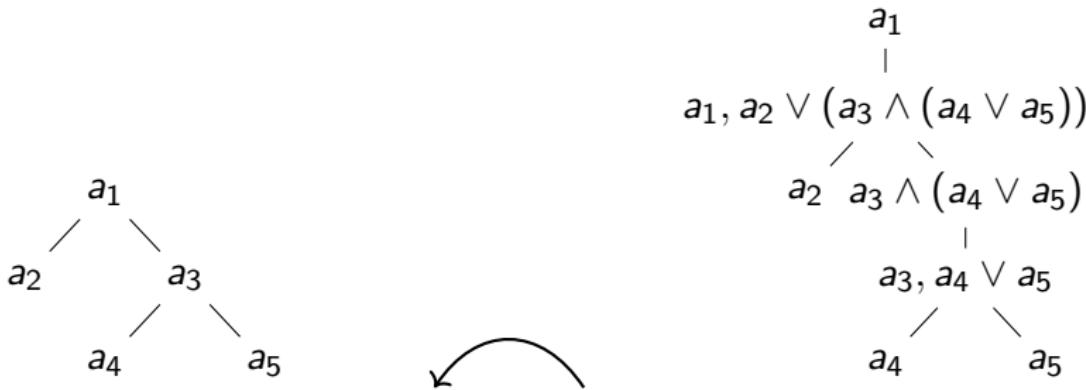
$$\varphi = a_1 \wedge (a_2 \vee (a_3 \wedge (\dots (a_n))))$$

Notre but en faisant une restriction du problème est:

- De mieux comprendre les avantages de cette méthode (dans quelle type de formule la méthode est-elle meilleur ?)
- De trouver des algorithmes polynomiales pour nos restrictions (si ce n'est possible, alors on améliorera aux maximum l'algorithme)

LP - réécriture

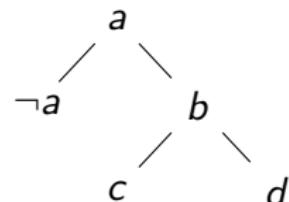
On re-écrit l'arbre induit par la méthode des tableaux d'une manière différente (ici pour $\varphi = a_1 \wedge (a_2 \vee (a_3 \wedge (a_4 \vee a_5)))$):



LP - Résolution

L'algorithme récursif consiste à faire ces analyses (en créant un dictionnaire stockant le "signe" des littéraux):

- 1 On analyse le littéral droit, si il y a contradiction, l'arbre est fermé, sinon on ajoute éventuellement dans le dictionnaire le littéral
- 2 On analyse le littéral gauche, si il produit une contradiction, appel recursif plus profond dans l'arbre, sinon la formule est satisfiable



Le cas de base étant l'arrivée au bout du peigne.

LP - Preuves et Analyse

L'algorithme est en $\mathcal{O}(n)$, en supposant les opérations Hashtbl constantes.

LP - Preuves et Analyse

L'algorithme est en $\mathcal{O}(n)$, en supposant les opérations Hashtbl constantes.

- La correction (preuve faite) est assuré par l'invariant "Toutes les branches déjà traités sont fermés"
- La terminaison (preuve faite) est assuré simplement.

LP - Preuves et Analyse

L'algorithme est en $\mathcal{O}(n)$, en supposant les opérations Hashtbl constantes.

- La correction (preuve faite) est assuré par l'invariant "Toutes les branches déjà traités sont fermés"
- La terminaison (preuve faite) est assuré simplement.

On crée une base de donnée de 100 formules de forme alternée et on fait tourner Quine et notre algorithme dessus.

- Alternée: 0.000493s
- Méthode des tableaux: ...
- Quine (avec conversion en CNF): 0.025874s
- Quine (sans conversion en CNF): 0.018691s

LP - Conclusion

Remarques:

- Bien entendu, il n'est pas possible de convertir toutes les formules en Alternée.
- On a donc trouvé un exemple de type de formule qui est facile à analyser par méthode des tableaux.

Nous allons maintenant élargir notre étude et s'intéresser à la logique linéaire temporelle (LTL).

LTL - Intro 1

Definition (Formule de la LTL)

On définit F l'ensemble des formules de la LTL inductivement par:

- $p \in AP \implies p \in F$
- si ψ et ϕ sont des formules de LTL alors
 $\neg\psi, \phi \vee \psi, X\psi, \phi U \psi, F\psi, G\psi$ sont des formules de LTL

AP un ensemble fini de variables propositionnelles.

Definition (Opérateurs X, U, F, G)

- $X\phi$: ϕ doit être satisfaite dans l'état suivant (neXt)
- $\psi U \phi$: ψ doit être satisfaite dans tous les états jusqu'à un état où ϕ est satisfait (Until)
- $F\phi$: ϕ doit être satisfaite une fois plus tard (Finally)
- $G\phi$: ψ doit être satisfaite dans tous les états futurs (Globally)

LTL - Intro 2

Definition (Monde)

On définit un tel objet comme $\omega := w_0, w_1, \dots$ une suite infinie d'état. On écrira $\omega^i := w_i, w_{i+1}, \dots$ un suffixe de ω .

Soit $v : \omega \times F \rightarrow \{T, F\}$ une fonction de valuation.

Definition (Satisfaction d'un monde ($\omega \models f$))

- $\omega \models a \Leftrightarrow v(w_0, a) = T$ si a atomique
- $\omega \models \neg f$ si $\neg(\omega \models f)$
- $\omega \models f \vee g$ si $\omega \models f$ ou $\omega \models g$
- $\omega \models \mathbf{X}f$ si $\omega^1 \models f$
- $\omega \models f \mathbf{U} g$ si $\omega \models g$ ou $\omega \models f \wedge \mathbf{X}(f \mathbf{U} g)$
- $\omega \models \mathbf{F}f$ si $\exists i, \omega^i \models f$
- $\omega \models \mathbf{G}f$ si $\forall i, \omega^i \models f$

LTL - Intro 3

On peut trouver plusieurs applications à la LTL:

- Preuve de programme concurrentiel
- Raisonnement sur des circuits intégrés
- Raisonnement sur les protocoles de communications

Toutes ces applications s'inscrivent dans le cadre de la vérification de modèles qui est une méthode permettant de montrer la correction de systèmes informatiques complexes.

Definition (Verification de modèle (Model Checking))

Technique automatique qui étant donné un modèle à nombre d'état fini d'un système et des propriétés formelles, vérifie systématiquement si ces propriétés restent vrai pour ce modèle

LTL - Vérification de Modèle

Ici, nous nous intéresserons pas aux techniques utilisés dans le domaine. On retiendra seulement que la méthode des tableaux est très utilisé dans la logique modale qui englobe la LTL et que elle est particulièrement utile pour certaines techniques de vérifications de modèles.

CF par Exemple: A. Pnueli and L. Zuck. Probabilistic verification by tableaux. In 1st Annual Symposium on Logic in Computer Science (LICS), pages 322-331. IEEE Computer Society Press, 1986
Ainsi, nous allons étudier un exemple de formule de la LTL et étudier sa résolution par méthode des tableaux.

LTL - Implementation utilisée

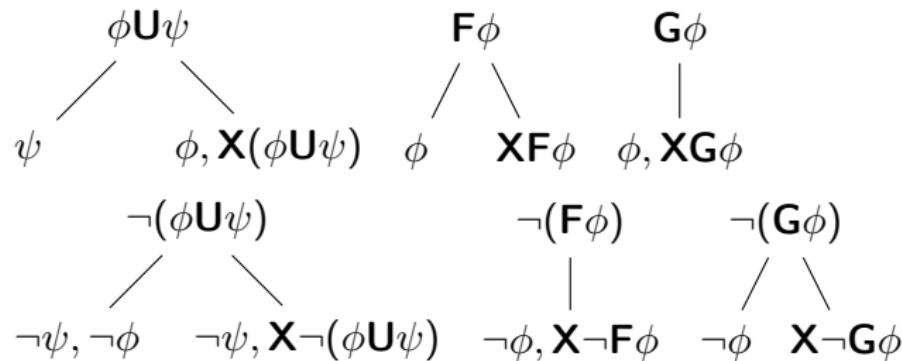
En réalité, la version de la méthode des tableaux que je présente en LTL est celle de Mark Reynolds, qui est une approche "tree-style" plus simple que les autres implémentations, mais qui reste correct. Faire une bonne implémentation d'une telle méthode reste assez fastidieux et il n'est pas astucieux ici de créer nous même une implémentation. C'est pour ça que l'on va s'aider de l'implémentation de Anatole Leterrier trouvable ici:

<https://github.com/anatcaramba/LTL-SAT-Solver-by-Reynolds-Tableaux>

Nous allons maintenant étudier une application de la méthode des tableaux.

LTL - Méthode des tableaux 1

On rajoute donc les règles suivantes:



En plus on rajoute des règles supplémentaires:

Definition

On rappelle **X**-eventualité les formules de la forme $\mathbf{X}(a \mathbf{U} b)$ et $\mathbf{X}\mathbf{F}b$

On notera $x \leq y$ si y est descendant de x (resp $<$ pour descendant strict)

LTL - Méthode des tableaux 2

- **LOOP:** Si un noeud v a un ancêtre strict u tq $\Gamma_v \subseteq \Gamma_u$ et pour chaque $\mathbf{X}(a \mathbf{U} b)$ ou $\mathbf{XF}b$ dans Γ_u , il existe un noeud w tq $b \in \Gamma_w$ et $u < w \leq v$ alors la formule de départ est satisfiable.
- **PRUNE:** Si $u < v < w$ et ont les mêmes hypothèses Γ et que pour chaque $\mathbf{X}(a \mathbf{U} b)$ ou $\mathbf{XF}b$ dans Γ , si on a x tq $b \in \Gamma_x$ et $v < x \leq w$ alors $\exists y, b \in \Gamma_y$ et $u < y \leq v$ et w ferme sa branche.
- **PRUNE₀:** Si $u < v$ et ont les mêmes hypothèses Γ et que il y a au moins une formule X-eventuelle, mais pas de formule de la forme $\mathbf{X}(a \mathbf{U} b)$ ou $\mathbf{XF}b$ dans Γ où $\exists x, b \in \Gamma_x$ et $u < x \leq v$ alors v ferme sa branche.
- **TRANSITION:** Si aucune des autres règles ne s'applique, alors un noeud étiqueté par Γ peut avoir un enfant dont l'étiquette est $\Delta = \{a | Xa \in \Gamma\} \cup \{\neg a | \neg Xa \in \Gamma\}$

LTL- Introduction de l'Etude

Remarques:

- Grâce à toutes ces règles, on peut montrer qu'un tel algorithme de résolution de SAT est complet et correct (il renvoie vrai si la formule est satisfiable et réciproquement).
- On sait que typiquement, les propriétés que doivent satisfaire notre modèle sont modélisées par des formules logiques en LTL.

Ainsi nous allons nous intéresser à la modélisation d'un objet technologique en LTL, en observant les hypothèses, que l'on fait etc...

Enfin, on pourra appliquer la méthode des tableaux pour conclure sur le bon fonctionnement de tout!

Code - Méthode des tableaux classique 1

```
type prop = | Var of string | Not of prop | And of prop * prop | Or  of prop *
            prop
(* Une branche c'est une liste de formule avec un signe *)
type branch = (bool * prop) list

let is_literal = function
  | (true, Var _) -> true
  | (false, Var _) -> true
  | (true, Not (Var _)) -> true
  | (false, Not (Var _)) -> true
  | _ -> false

(* Check les contradictions *)
let branch_closed (br : branch) : bool =
  let pos = Hashtbl.create 16 in
  let neg = Hashtbl.create 16 in
  let record = function
    | (true, Var v) -> Hashtbl.replace pos v true
    | (false, Var v) -> Hashtbl.replace neg v true
    | (true, Not (Var v)) -> Hashtbl.replace neg v true
    | (false, Not (Var v)) -> Hashtbl.replace pos v true
    | _ -> ()
  in
  List.iter record br;
  let closed = ref false in
  Hashtbl.iter (fun v _ -> if (Hashtbl.mem pos v) && (Hashtbl.mem neg v) then
    closed := true) pos;
  !closed
```

Code - Méthode des tableaux classique 2

```
(* La décomposition usuelle faites durant la méthode des tableaux *)
let decompose_once (br : branch) : branch list option =
    let rec find_nonlit acc = function
        | [] -> None
        | x :: xs ->
            if is_literal x then find_nonlit (x::acc) xs
            else Some (List.rev acc, x, xs)
    in
    match find_nonlit [] br with
    | None -> None
    | Some (left, (sign, form), right) ->
        let rest = left @ right in
        let mk b p = (b, p) in
        (match sign, form with
        | true, And (a,b) ->
            Some [ (mk true a) :: (mk true b) :: rest ]
        | false, Or (a,b) ->
            Some [ (mk false a) :: (mk false b) :: rest ]
        | true, Not a ->
            Some [ (mk false a) :: rest; (mk true b)::rest ]
        | false, And (a,b) ->
            Some [ (mk false a)::rest; (mk false b)::rest ]
        | true, Or (a,b) ->
            Some [ (mk true a) :: rest ]
        | false, Not a ->
            Some [ (mk false a) :: rest ]
        | _, _ -> None)
```

Code - Méthode des tableaux classique 3

```
(* La Methode des Tableaux en soit *)
let satisfiable (phi : prop) : bool =
let initial_branch = [ (true, phi) ] in
let rec explore_stack stack =
  match stack with
  | [] -> false
  | br :: rest ->
    if branch_closed br then explore_stack rest else
    match decompose_once br with
    | None -> true
    | Some new_branches -> explore_stack (new_branches @ rest)
in explore_stack [ initial_branch ]
```

Code - Alternée 1

```
type formula =
| Atom of (string* bool)
| And of (string*bool) * formula
| Or of (string*bool) * formula

type branch =
| Empty
| Node of (formula option * formula * branch);;

let extract (f:formula option) = match f with
| None -> Atom("none", false)
| Some t -> t

let rec print_formula (f:formula) = match f with
| Atom(s, b) -> if b then print_string s else print_string "Not";
    print_string s;
| And ((f, b),g) -> if b then print_string f else print_string "Not";
    print_string f;print_string "And";print_formula g
| Or ((f,b),g) -> if b then print_string f else print_string "Not";
    print_string f;print_string "Or";print_formula g;;

let rec print_branches (b:branch) =
print_string "[";
match b with
| Empty -> ()
| Node(a1, a2, b) -> print_formula@@extract a1;print_string ",";
    print_formula a2;print_branches b;
print_string "]";;
```

Code - Alternée 2

```
let rec formula2branch (f:formula) : branch = match f with
| And(a, Or(b, Atom(c))) -> Node(Some(Atom b), Atom a, Node(None, Atom(c),
    Empty))
| And(a, Or(b, c)) -> Node(Some(Atom b), Atom a, formula2branch c)
| And(a, Atom(b)) -> Node(Some (Atom b), Atom a, Empty)
| _ -> failwith "Pas d'alternée"

let has_cycle (br:branch) : bool =
let rec aux (br:branch) (d:(string,bool) Hashtbl.t) : bool = match br with
| Node(None, Atom (f, b), Empty) ->
    if Hashtbl.mem d f then
        Hashtbl.find d f = b
    else
        true
| Node(Some(Atom(fg, bg)), Atom (fd, bd), Empty) ->
    if Hashtbl.mem d fd then
        if Hashtbl.find d fd = bd then
            not @@ Hashtbl.mem d fg && Hashtbl.find d fg <> bg
        else
            false
    else(
        Hashtbl.add d fd bd;
        not @@ Hashtbl.mem d fg && Hashtbl.find d fg <> bg)
| Node(Some (Atom (fg, bg)), Atom (fd, bd), nb) ->
    if Hashtbl.mem d fd then
        if Hashtbl.find d fd <> bd then
```

Code - Alternée 3

```
false
else
  if Hashtbl.mem d fg then
    if Hashtbl.find d fg = bg then
      true
    else
      aux nb d
  else
    true
else
  (Hashtbl.add d fd bd;
  if Hashtbl.mem d fg then
    if Hashtbl.find d fg = bg then
      true
    else
      aux nb d
  else
    true)
| _ -> failwith "Pas d'alternee"
in aux br (Hashtbl.create 100);;

let is_satisfiable (f:formula) : bool = let b = formula2branch f in has_cycle b
;;
```