

TIPE 25/26 - Cycles et Boucles

Méthode des tableaux : Optimisation pour les clauses de Horn

GIL Dorian

Sommaire

- 1 Présentation Méthode
- 2 Exemple d'Application
- 3 Implémentation en OCaml
- 4 Objectifs futurs

Présentation

Definition (Méthode des tableaux)

Algorithme pour prouver une assertion ϕ ayant pour hypothèse (H_n) en montrant que $\neg\phi$ est insatisfaisable.

Definition (Clause de Horn)

Clause (i.e con/disjonction de littéraux) comportant au plus un littéral positif

Présentation

Definition (Méthode des tableaux)

Algorithme pour prouver une assertion ϕ ayant pour hypothèse (H_n) en montrant que $\neg\phi$ est insatisfaisable.

Definition (Clause de Horn)

Clause (i.e con/disjonction de littéraux) comportant au plus un littéral positif

- On place $\neg\phi$ et ses hypothèses dans la racine.
- On applique des règles (R_x) à chaque formule en bout d'arbre qui sont développables
- Si on trouve a et $\neg a$ dans l'arbre (un *cycle*), alors ϕ est vrai

Présentation

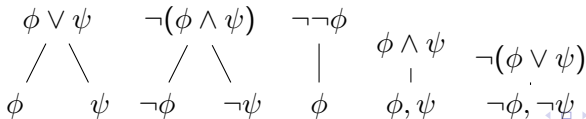
Definition (Méthode des tableaux)

Algorithme pour prouver une assertion ϕ ayant pour hypothèse (H_n) en montrant que $\neg\phi$ est insatisfaisable.

Definition (Clause de Horn)

Clause (i.e con/disjonction de littéraux) comportant au plus un littéral positif

- On place $\neg\phi$ et ses hypothèses dans la racine.
- On applique des règles (R_x) à chaque formule en bout d'arbre qui sont développables
- Si on trouve a et $\neg a$ dans l'arbre (un *cycle*), alors ϕ est vrai



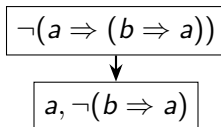
Exemple

Formule: $a \Rightarrow (b \Rightarrow a)$

$$\neg(a \Rightarrow (b \Rightarrow a))$$

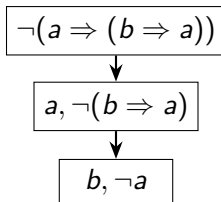
Exemple

Formule: $a \Rightarrow (b \Rightarrow a)$



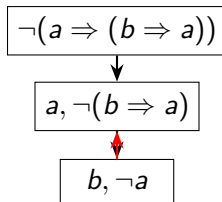
Exemple

Formule: $a \Rightarrow (b \Rightarrow a)$



Exemple

Formule: $a \Rightarrow (b \Rightarrow a)$



Implémentation 1

```
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;;
```

Première implémentation en OCaml, sans les hypothèses

Implémentation 2

```
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];;
```

Première implémentation en OCaml, sans les hypothèses

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.
- Trouver et prouver des optimisations.

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.
- Trouver et prouver des optimisations.
- Implémenter et commenter les résultats de l'optimisation.

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.
- Trouver et prouver des optimisations.
- Implémenter et commenter les résultats de l'optimisation.
- Faire de même en logique du première ordre OU continuer à trouver des optimisations dans la logique propositionnelle.

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.
- Trouver et prouver des optimisations.
- Implémenter et commenter les résultats de l'optimisation.
- Faire de même en logique du première ordre OU continuer à trouver des optimisations dans la logique propositionnelle.

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.
- Trouver et prouver des optimisations.
- Implémenter et commenter les résultats de l'optimisation.
- Faire de même en logique du première ordre OU continuer à trouver des optimisations dans la logique propositionnelle.

Sur le court terme :

- Ajouter les hypothèses à l'implémentation.

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.
- Trouver et prouver des optimisations.
- Implémenter et commenter les résultats de l'optimisation.
- Faire de même en logique du première ordre OU continuer à trouver des optimisations dans la logique propositionnelle.

Sur le court terme :

- Ajouter les hypothèses à l'implémentation.
- Faire les preuves de correction, terminaison pour toutes formules.

Ce qui est à faire

Mon but sur le long terme

- Ajouter les hypothèses à l'implémentation et faire les preuves de correction, terminaison pour toutes formules.
- Trouver et prouver des optimisations.
- Implémenter et commenter les résultats de l'optimisation.
- Faire de même en logique du première ordre OU continuer à trouver des optimisations dans la logique propositionnelle.

Sur le court terme :

- Ajouter les hypothèses à l'implémentation.
- Faire les preuves de correction, terminaison pour toutes formules.
- Commencer la recherche d'optimisation