

TD 4. Modélisation et DPLL

Exercice 1. DPLL à la main

Nous rappelons que l'algorithme DPLL décide la satisfiabilité d'une formule en CNF par l'application des règles suivantes à l'ensemble de ses clauses, en donnant la priorité aux règles (unit) et (pure) (car elles ne demandent jamais un retour en arrière) avant de brancher avec (split_P) ou ($\text{split}_{\neg P}$).

| | | |
|-----------------------------|--|-----------------------------|
| (unit) | $F \cup \{\{\ell\}\} \rightarrow_{\text{dpll}} F[\top/\ell]$ | |
| (pure) | $F \rightarrow_{\text{dpll}} F[\top/\ell]$ | où $\ell \in \text{Pur}(F)$ |
| (split_P) | $F \rightarrow_{\text{dpll}} F[\top/P]$ | où $P \in \text{fp}(F)$ |
| ($\text{split}_{\neg P}$) | $F \rightarrow_{\text{dpll}} F[\top/\neg P]$ | où $P \in \text{fp}(F)$ |

Appliquer l'algorithme DPLL à la formule

$$\begin{aligned} \varphi = & (P \vee Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (P \vee \neg S) \\ & \wedge (\neg Q \vee \neg R \vee \neg S) \wedge (\neg P \vee \neg Q \vee R) \wedge (T \vee U) \\ & \wedge (T \vee \neg U) \wedge (Q \vee \neg T) \wedge (\neg R \vee \neg T). \end{aligned}$$

Exercice 2. Le format DIMACS

Le but de cet exercice est de comprendre le format DIMACS qui est utilisé par les solveurs SAT, et par l'algorithme que vous allez implémenter dans le mini-projet 1.

Nous rappelons que le type `formula` peut être défini en OCaml par :

```

1 type formula =
2   | Prop of int
3   | Neg of formula
4   | And of formula * formula
5   | Or of formula * formula

```

On supposera dans cet exercice que les variables propositionnelles sont représentées par des entiers *strictement positifs*. Nous rappelons aussi qu'une formule en forme clausale peut être représentée comme une liste de listes d'entiers : par exemple, la formule en CNF

$$(P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee Q_1) \wedge (\neg Q_2 \vee \neg P_3)$$

peut être représenté par la liste de listes $[[1;2;3]; [-1;4]; [-5;-3]]$, où chaque proposition P_i est représentée par l'entier i et chaque Q_i est représentée par l'entier $i + 3$. On remarque que si un entier strictement positif n représente une proposition Q , alors $-n$ représente le littéral $\neg Q$. C'est l'essentiel de la représentation au format DIMACS des formules en forme clausale.

Dans cet exercice, nous souhaitons écrire une fonction `dimacs_of_cnf` qui renvoie la liste de listes d'entiers qui correspond à une formule donnée sous forme clausale.

(a) Donnez une liste de listes d'entiers qui représente la formule en CNF $\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3$, où

$$\varphi_1 = P_1 \vee \neg P_2 \vee \neg P_3, \quad \varphi_2 = \neg P_1 \vee P_2 \vee \neg P_3, \quad \varphi_3 = \neg P_1 \vee \neg P_2 \vee P_3.$$

(b) Écrivez une fonction (en OCaml) `is_clause : formula -> bool` qui retourne `true` si et seulement si la formule donnée en entrée est une clause, c'est-à-dire une disjonction de littéraux. Par exemple, avec les notations de la question précédente, `is_clause phi1` retourne `true` mais `is_clause phi` retourne `false`.

- (c) Écrivez une fonction `is_cnf : formula -> bool` qui retourne `true` si et seulement si la formule donnée en entrée est en CNF, c'est-à-dire une conjonction de clauses. Par exemple, `is_cnf phi` retourne `true` mais `is_cnf (Neg phi1)` retourne `false`.
- (d) Écrivez une fonction `list_of_clause : formula -> int list` qui retourne la liste d'entiers correspondante à la clause donnée en argument, ou échoue avec le message "not a clause" si la formule donnée en entrée n'est pas une clause. Par exemple, `list_of_clause phi2` retourne la liste `[1; -2; 3]` (ou une permutation).
- Indications.* On peut utiliser `failwith "not a clause"` pour les échecs. Pour concaténer deux listes `lst1` et `lst2`, on peut écrire `lst1 @ lst2`.
- (e) Écrivez une fonction `clauses_of_cnf : formula -> formula list` qui retourne la liste de clauses d'une formule en CNF, ou échoue avec le message "not in CNF". Par exemple, `clauses_of_cnf phi` retourne la liste `[phi1; phi2; phi3]` (ou une permutation).
- (f) Écrivez une fonction `dimacs_of_cnf : formula -> int list list option` qui retourne `Some lst` si la formule donnée en entrée est en CNF, où `lst` est la liste de listes d'entiers qui correspondent à la formule, et qui retourne `None` autrement.

Exercice 3. Principe des tiroirs

Le principe des tiroirs affirme que k pigeons peuvent être mis dans n tiroirs, de manière à ce que chaque pigeon soit dans un tiroir distinct, si et seulement si $k \leq n$.

- Pour k, n fixés, donner un ensemble de propositions qui permettent de modéliser le placement des k pigeons dans les n tiroirs.
- Donner une formule en CNF modélisant le fait que chaque pigeon est dans au moins l'un des tiroirs.
- Donner une formule en CNF modélisant le fait que chaque tiroir contient au plus un pigeon.
- Utiliser l'algorithme DPLL pour montrer que 2 pigeons peuvent se répartir entre 2 tiroirs. On ne prendra pas en compte dans la modélisation le fait qu'un pigeon ne peut pas être dans deux tiroirs à la fois.
- (Bonus) Utiliser l'algorithme DPLL pour montrer que 3 pigeons ne peuvent pas se répartir entre 2 tiroirs.

Exercice 4. Modélisation par recouvrement exact et résolution par DPLL

Le problème de recouvrement exact peut être décrit abstraitelement de la manière suivante : étant donné une matrice de 0s et de 1s, existe-t-il un ensemble de lignes contenant exactement un 1 dans chaque colonne ?

- La matrice suivante a-t-elle un tel ensemble de lignes ?

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

- Le premier intérêt du problème de recouvrement exact est qu'il est facile à coder dans SAT-CNF. Étant donné un problème de recouvrement exact, expliquer comment construire une formule en CNF φ telle que les interprétations qui satisfont φ correspondent aux solutions du problème de recouvrement exact.
- Appliquer cette méthode à la matrice ci-dessus et appliquer DPLL à la CNF obtenue.
- Le second intérêt du problème de recouvrement exact est qu'il permet d'encoder simplement un grand nombre de problèmes combinatoires naturels. Étant donnée une grille de Sudoku 9×9 à résoudre, expliquer comment construire un problème de recouvrement exact dont les solutions correspondent à celles de la grille.

Ey1

Exercice 1. DPLL à la main

Nous rappelons que l'algorithme DPLL décide la satisfiabilité d'une formule en CNF par l'application des règles suivantes à l'ensemble de ses clauses, en donnant la priorité aux règles (unit) et (pure) (car elles ne demandent jamais un retour en arrière) avant de brancher avec (split_P) ou ($\text{split}_{\neg P}$).

| | | |
|-----------------------------|--|-----------------------------|
| (unit) | $F \cup \{\{\ell\}\} \rightarrow_{\text{dpll}} F[\top/\ell]$ | |
| (pure) | $F \rightarrow_{\text{dpll}} F[\top/\ell]$ | où $\ell \in \text{Pur}(F)$ |
| (split_P) | $F \rightarrow_{\text{dpll}} F[\top/P]$ | où $P \in \text{fp}(F)$ |
| ($\text{split}_{\neg P}$) | $F \rightarrow_{\text{dpll}} F[\top/\neg P]$ | où $P \in \text{fp}(F)$ |

Appliquer l'algorithme DPLL à la formule

$$\begin{aligned}\varphi = & (P \vee Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (P \vee \neg S) \\ & \wedge (\neg Q \vee \neg R \vee \neg S) \wedge (\neg P \vee \neg Q \vee R) \wedge (T \vee U) \\ & \wedge (T \vee \neg U) \wedge (Q \vee \neg T) \wedge (\neg R \vee \neg T).\end{aligned}$$

$$\begin{aligned}\varphi = & (P \vee Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (P \vee \neg S) \\ & \wedge (\neg Q \vee \neg R \vee \neg S) \wedge (\neg P \vee \neg Q \vee R) \wedge (T \vee U) \\ & \wedge (T \vee \neg U) \wedge (Q \vee \neg T) \wedge (\neg R \vee \neg T)\end{aligned}$$

P Q R U T S

formule clause :

$$\{ \{P, Q, R\}, \{P, \neg Q, \neg R\}, \{P, \neg S\}, \{\neg Q, \neg R, \neg S\}, \\ \{\neg P, \neg Q, R\}, \{T, U\}, \{\neg T, \neg U\}, \{Q, \neg T\}, \{\neg R, \neg T\}$$

$\rightarrow \neg S$: pur il apparaît $\neg S$ seulement

$$\begin{array}{ll} \{ \{P, Q, R\}, \{P, \neg Q, \neg R\}, \{\neg P, \neg Q, R\}, \{T, U\}, \{\neg T, \neg U\}, \{Q, \neg T\}, \{\neg R, \neg T\} \} & T=1 \\ & T=0 \\ \{ \{P, Q, R\}, \{P, \neg Q, \neg R\}, \{\neg P, \neg Q, R\}, \{T, U\}, \{\neg T, \neg U\} \} & \{ \{P, Q, R\}, \{P, \neg Q, \neg R\}, \{\neg P, \neg Q, R\}, \{T, U\}, \{\neg T, \neg U\} \} \end{array}$$

$Q = 1$ unitaire

$U = \text{unitaire}$

$$\{(P, \neg R), (\neg P, R), (\neg R)\}$$

$$\{(P, Q, R), (\neg P, \neg Q, \neg R), (\neg P, \neg Q, R)\}$$

, \perp

x contradiction

$$\{U\} \wedge \neg_U U$$

$\neg R$ unitaire

$$\{\neg P\}$$

$\neg P$ unitaire

\emptyset

φ satisfiable:

$$I = [0/P, 1/Q, 0/R, 0/S, 1/T, 0/U]$$

excl:

$$\text{d}\varphi_1 = P_1 \vee \neg P_2 \vee \neg P_3$$

$$\varphi_2 = \neg P_1 \vee P_2 \vee \neg P_3$$

$$\varphi_3 = \neg P_1 \vee \neg P_2 \vee P_3$$

$$\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \quad [(1; -2; -3); (-1; 2; 3)] \\ , [-1; -2; 3]]$$

b) let rec is_clause phi =
 match phi with :
 | Prop b → true
 | Neg (Prop b) → true
 | Or (phi1, phi2) → (is_clause phi1) && (is_clause phi2)
 | _ → false

$$\varphi = (P_1 \vee P_2 \vee P_3) \wedge (\neg P_1 \vee Q_1) \wedge (\neg Q_2 \vee \neg P_3) \quad \text{Non} \rightarrow \text{false}$$

$$\varphi_1 = P_1 \vee \neg P_2 \vee \neg P_3, \quad \varphi_2 = \neg P_1 \vee P_2 \vee \neg P_3, \quad \varphi_3 = \neg P_1 \vee \neg P_2 \vee P_3. \quad \text{Oui true}$$

c) let rec is_cnf phi =

match phi with

| And (phi1, phi2) → (is_cnf phi1) && (is_cnf phi2)
 | _ → is_clause phi

d) let rec list_of_clause cl =

match cl with

| Prop p → [p]

| Neg (Prop p) → [-p]

| Or (Prop p) → (list_of_clause phi1) @
 (list_of_clause phi2)

concatener ↑

| _ → failwith "not a clause"

e)

- (e) Écrivez une fonction clauses_of_cnf : formula → formula list qui retourne la liste de clauses d'une formule en CNF, ou échoue avec le message "not in CNF". Par exemple, clauses_of_cnf phi retourne la liste [phi1; phi2; phi3] (ou une permutation).
- (f) Écrivez une fonction dimacs_of_cnf : formula → int list list option qui retourne Some lst si la formule donnée en entrée est en CNF, où lst est la liste de listes d'entiers qui correspond à la formule, et qui retourne None autrement.

$\text{C}_\phi \text{ let rec clause_of_cnf } \phi =$

is_clause ϕ then [ϕ]

else match ϕ with

| And (ϕ_1, ϕ_2) \rightarrow (clauses_of_cnf ϕ_1)

@ (clauses_of_cnf ϕ_2)

| - fail with "not in cnf"

Gérer des volets qui peuvent être présents (Some) ou absents (None)

$f_\phi \text{ let format_dianacs } \phi =$

if is_cnf ϕ then Some (List map list_of_clauses
(clauses_of_cnf ϕ))

else None

Dianacs : format de fichier qui est utilisé pour représenter des problèmes logiques exprimés en CNF