DS5 (éléments de réponse)

Exercice 1

Question 1.
□ **1.1.** On raisonne par cas sur le nombre de littéraux valides parmi l_1, l_2 et l_3.

- Si les trois sont valides, alors les clauses $l_1, l_2, l_3, l_1 \vee \neg x, l_2 \vee \neg x$ et $l_3 \vee \neg x$ sont valides et les clauses $\overline{l_1} \vee \overline{l_2}, \overline{l_2} \vee \overline{l_3}$ et $\overline{l_3} \vee \overline{l_1}$ sont invalides, indépendamment de x. Choisir $\vee$ pour x valide une septième clause, et choisir F ne permet rien de plus.
- Si deux sont valides. Par symétrie, on suppose qu'il s'agit de l_1 et l_2. Alors les six clauses $l_1, l_2, \overline{l_2} \vee \overline{l_3}$, $\overline{l_3} \vee \overline{l_1}, l_1 \vee \neg x, l_2 \vee \neg x$ sont valides indépendamment de x. Selon le choix d'une valeur pour x, on valide en plus soit x, soit $l_3 \vee \neg x$, c'est-à-dire 7 au total dans tous les cas.
- Si un seul est valide. Par symétrie, on suppose qu'il s'agit de l_1. Alors les cinq clauses $l_1, \overline{l_1} \vee \overline{l_2}, \overline{l_2} \vee \overline{l_3}, \overline{l_3} \vee \overline{l_1}$ et $l_1 \vee \neg x$ sont valides indépendamment de x. Selon le choix d'une valeur pour x, on valide en plus soit x, soit $l_2 \vee \neg x$ et $l_3 \vee \neg x$, c'est-à-dire 7 au maximum.

□ **1.2.** Si aucune n'est valide, alors les clauses $\overline{l_1} \vee \overline{l_2}, \overline{l_2} \vee \overline{l_3}, \overline{l_1} \vee \overline{l_3}$ sont valides, et les clauses l_1, l_2 et l_3 ne le sont pas. Selon le choix d'une valeur pour x, on valide en plus soit x, soit $l_1 \vee \neg x, l_2 \vee \neg x$ et $l_3 \vee \neg x$, c'est-à-dire 6 au maximum.

Question 2. Pour chaque clause $l_1 \wedge l_2 \wedge l_3$ de notre formule φ, on construit un groupe de dix clauses tel qu'à la question précédente, avec x une variable non encore utilisée. Dans le cas d'une clause l_1 unaire ou d'une clause $l_1 \vee l_2$ binaire, on complète d'abord en $l_1 \wedge l_1 \wedge l_1$ (resp. $l_1 \wedge l_1 \wedge l_2$) puis on construit le même groupe. On obtient alors une formule φ' comportant $10m$ clauses, et on fixe le seuil $k = 7m$.

Si la formule φ est satisfiable, alors il existe une valuation v pour φ, pour laquelle au moins un littéral est valide dans chaque clause. On peut étendre cette valuation pour φ' de sorte que 7 clauses soit valides dans chaque groupe, d'où $7m$ clauses valides au total. À l'inverse, supposons qu'il existe une valuation v' satisfaisant $7m$ clauses de φ'. On a vu que seules 7 clauses au maximum pouvaient être simultanément satisfaites dans chacun des m groupes. La valuation v' satisfait donc exactement 7 clauses par groupe. Par l'analyse de la question précédente, on sait que cela n'est possible que pour une valuation satisfiant la clause de φ correspondante. Donc toutes les clauses de φ sont satisfaites par v', et la formule est bien satisfiable

Algorithme probabiliste

Question 3. Considérons une clause C avec k littéraux indépendants $l_1, l_2, \dots, l_k$. Cette clause n'est fausse que si tous ses littéraux le sont, événement dont la probabilité est divisée par deux pour chaque littéral supplémentaire :

$$\begin{aligned} P(C \text{ est fausse }) &= P\left(\forall i \in [\![1;k]\!], l_i \text{ n'est pas satisfaite }\right) \\ &= \prod_{i=1}^{k} \frac{1}{2} && \text{par indépendance des événements} \\ &= \frac{1}{2^k} \end{aligned}$$

La probabilité que cette clause soit satisfaite par notre valuation aléatoire est donc au contraire $1 - \frac{1}{2^k}$, et augmente avec le nombre k de littéraux. Ainsi, si φ ne contient que des clauses avec au moins k littéraux, l'espérance du nombre de clauses satisfaites par une valuation aléatoire est :

$$\mathbb{E}(\text{sat}(v, \varphi)) = \mathbb{E}\left(\sum_{j=1}^{m} \mathbb{1}_{\text{clause } C_j \text{ satisfaite}}\right) = \sum_{j=1}^{m} P\left(C_j \text{ est satisfaite }\right) \geqslant m\left(1 - \frac{1}{2^k}\right)$$

Question 4. Sans restriction sur le nombre de littéraux par clause, on peut appliquer le théorème précédent avec $k = 1$, et minorer l'espérance par $\frac{m}{2}$. Ainsi, prendre une valuation aléatoire faite de n tirages indépendants donne de bonnes chances de satisfaire un nombre de clauses au moins égal à la moitié du nombre maximal de clauses satisfiables simultanément (et même plus, si la plupart des clauses ne sont pas trop petites).

Question 5.
□ **5.1.** L'algorithme n'est pas une 1/2-approximation de Max-2Sat. Si on n'a pas de chance, l'algorithme peut renvoyer une valuation qui ne satisfait aucune clause alors que toutes sont satisfiables simultanément. Par exemple, la formule $\varphi = x_1 \wedge x_2$ contenant deux clauses à une variable chacune. L'algorithme peut associer toutes les variables à `false`, auquel cas aucune clause n'est satisfaite, alors qu'on peut en satisfaire 2.
□ **5.2.** On commence par définir deux fonctions `var_max` et `n_var`.

```
let rec var_max lst = match lst with
  | [] -> 0
  | litt :: c -> max (max litt (-litt)) (var_max c)

let rec n_var phi = match phi with
  | [] -> 0
  | c :: phi -> max (var_max c) (n_var phi)

let maxSat_proba phi =
  let n = n_var phi in
  Array.init (n+1) (fun _ -> Random.int 2)
```

Algorithme d'approximation pour MaxSat

Question 6.

```
let maxSat_approx phi =
  let n = n_var phi in
  let v = Array.make (n + 1) false in
  let k = List.fold_left max 0 (List.map List.length phi) in
  let b = 1 lsl k in (* shift 1 de k bits vers la gauche, i.e. b = 2^k *)
  (* on pourrait le calculer par exponentiation rapide *)
  for i = 1 to n do
    (* probabilité conditionnelle de satisfaction d'une clause *)
    let expect_cl cl =
      let sat_literal x = abs x <= i && x > 0 = v.(abs x) in
      if List.exists sat_literal cl then b
      else let cl' = List.filter (fun x -> abs x > i) cl in
        b - b / (1 lsl List.length cl') (* idem, c'est 2^nb_litteraux*)
    in
    (* espérance conditionnelle *)
    let expect () = List.fold_left (+) 0
    (List.map expect_cl phi) in
    let exp_f = expect () in
    v.(i) <- true;
    let exp_t = expect () in
    if exp_f > exp_t then v.(i) <- false
  done;
  v
```

Question 7. En notant v_i la valuation partielle construite après i tours de boucle, on démontre d'abord que la propriété « $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i\right) \geqslant \mathbb{E}(\text{sat}(v,\varphi))$ » est un invariant de la boucle Pour de l'algorithme.

- **Initialisation** : À l'entrée de la boucle, la valuation partielle v_0 est vide et on a $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i\right) = \mathbb{E}(\text{sat}(v,\varphi))$.
- **Conservation** : Supposons que $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i\right) \geqslant \mathbb{E}(\text{sat}(v,\varphi))$. Au tour $i+1$, l'algorithme complète v_i en une valuation partielle v_{i+1} telle que :

$$\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_{i+1}\right) = \max\left(\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i, x_{i+1} = V\right), \mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i, x_{i+1} = F\right)\right)$$

  Or l'événement « $x_{i+1} = V$ » a une probabilité $P\left(x_{i+1} = V\right) = \frac{1}{2}$ indépendante de v_i, car on regarde un tirage aléatoire v sachant les valeurs fixées jusqu'à x_i incluse. Donc par linéarité de l'espérance, on a :

$$\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i\right) = \frac{1}{2}\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i, x_{i+1} = V\right) + \frac{1}{2}\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i, x_{i+1} = F\right)$$

  L'une des deux espérances conditionnelles $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i, x_{i+1} = V\right)$ ou $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i, x_{i+1} = F\right)$ est donc supérieure ou égale à $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i\right)$. Comme v_{i+1} est choisie telle que $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_{i+1}\right)$ soit la plus grande de ces deux valeurs, on peut conclure que :

$$\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_{i+1}\right) \geqslant \mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i\right) \geqslant \mathbb{E}(\text{sat}(v,\varphi))$$

Question 8. La propriété $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_i\right)$ étant un invariant, elle est encore vraie après le dernier tour de boucle. À ce moment-là, on a donc $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_n\right) \geqslant E(\text{sat}(v,\varphi))$, avec v_n une valuation donnant une valeur de vérité à chacune des n variables de φ. La valuation v_n étant totale, la valeur de chaque clause est définie, et l'espérance $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_n\right)$ est donc précisément le nombre de clauses satisfaites par v_n. Ceci conclut la démonstration puisque l'algorithme renvoie alors la valuation v_n, qui satisfait $\mathbb{E}\left(\text{sat}(v,\varphi) \mid v_n\right) \geqslant E(\text{sat}(v,\varphi))$ clauses de φ.

Exercice 2

Question 1.

□ **1.1.**

```
let minimize cmp eval lst =
  let rec trouve ((m,v) as c) = function
    | [] -> c
    | x' :: lst' -> let v' = eval x' in
                if cmp v' v
                then trouve (x',v') lst'
                else trouve c lst'
  in
  match lst with
    | [] -> raise (Invalid_argument "minimize: empty list")
    | x :: lst' -> trouve (x,eval x) lst'
;;
```

□ **1.2.**

```
let min (eval : 'a -> float) lst = minimize (prefix <) eval lst;;
let max (eval : 'a -> float) lst = minimize (prefix >) eval lst;;
```

Question 2.

□ **2.1.**

```
(* minmax
  eval : 'a -> float
  suiv : 'a -> 'a list
  dep : 'a
  renvoie le coup à jouer et son évaluation :
    'a (pos. choisie) * float profondeur 1. *)

let minmax eval suiv dep =
  let eval1 p = match eval p with
    | 0.0 -> 1.0
    | 1.0 -> 0.0
    | _ -> snd (min eval (suiv p))
  in
  max eval1 (suiv dep)
;;
```

□ **2.2.**

```
let minmax_n eval suiv n dep =
  let rec eval1 n p = match eval p with
    | 0.0 -> 1.0
    | 1.0 -> 0.0
    | _ -> if n = 0
         then snd (min eval (suiv p))
         else snd (min (fun p -> snd (minmax_rec (pred n) p)) (suiv p))
  and minmax_rec n p = match eval p with
    | (0.0 | 1.0) as e -> p, e
    | _ -> max (eval1 n) (suiv p)
  in
  minmax_rec n dep
;;

(* remarque : minmax = (minmax_n 0) *)
```

Question 3.

□ **3.1.**

```
type configuration == bool * int ;;
let (configuration_initiale : configuration) = true, 20 ;;

let affiche_config (b,n) =
  print_string (if b then "au joueur 1 : " else "au joueur 2 : ") ;
  for i = 1 to n do print_string "|" done ;
  print_newline ()
;;
```

□ **3.2.**

```
let (suiv : configuration -> configuration list) =
  fun (b,n) -> map (fun n' -> not b,n')
                   (match n with
                     | 0 -> []
                     | 1 -> [0]
                     | 2 -> [0 ; 1]
                     | n -> [n-3 ; n-2 ; n-1])
;;
```

□ **3.3.**

```
let (eval : configuration -> float) =
  fun (b,n) -> match n with
    | 0 -> 1.0
    | _ -> 0.5
;;
```

□ **3.4.**

```
let joue ((b,n) as c) = fst (minmax_n eval suiv (n/2+1) c) ;;
```

□ **3.5.**

```
let boucle coup ci =
  let rec boucle_rec ((b,n) as c) =
    if n = 0 then begin
      print_string (if b then "le joueur 1 gagne !" else "le joueur 2 gagne !")
   ;
      print_newline ()
    end
    else begin
      affiche_config c ;
      boucle_rec (coup c)
    end
  in
  boucle_rec ci
;;
```