DS MP : Option Informatique – Durée 4 heures

A. Arbres de décision

Un arbre de décision est un arbre binaire dans lequel :

— un nœud interne est associé à une variable, parmi un ensemble V de variables ;

— une feuille est associée à un booléen (vrai ou faux).

Si chaque variable de l'ensemble V reçoit une valeur booléenne, un tel arbre permet de prendre une décision en parcourant l'arbre :

— on part de la racine ;

— quand on arrive sur un nœud interne (racine comprise), on regarde quelle est la valeur de la variable associée au nœud : si elle vaut vrai on poursuit le parcours dans le sous-arbre gauche, sinon on poursuit le parcours dans le sous-arbre droit ;

— quand on arrive sur une feuille, le booléen associé constitue la décision.

Conventionnellement, on représente l'arête menant au sous-arbre pour le « cas vrai » en trait plein ; l'arête menant au sous-arbre « cas faux » en pointillés. Schématiquement, un arbre est structuré comme indiqué ci-dessous à gauche.

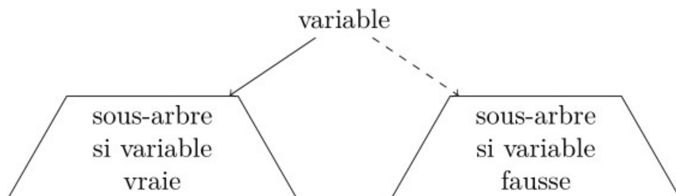

Figure 1. Schéma d'un arbre de décision.

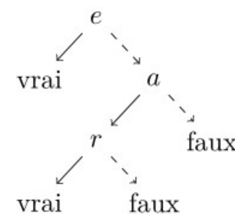

Figure 2. Arbre de décision associé à la formule $e \vee (a \wedge r)$.

Le schéma de droite ci-dessus illustre l'exemple : un module de cours est validé si l'examen est réussi (e), ou sinon, si l'étudiant a été assidu en cours (a) et qu'il réussit un examen de rattrapage (r). Cela revient à définir la validation du module par la formule logique $e \vee (a \wedge r)$.

Dans la suite, **on utilise une représentation impérative** (modifiable) des arbres de décision, que l'on codera par des tableaux. Plus précisément, on numérote les nœuds : la racine reçoit le numéro 0, et les autres nœuds sont numérotés arbitrairement par des entiers consécutifs à partir de 1 (il n'y a donc pas unicité de la représentation). On crée un tableau contenant autant de cases que de nœuds, indicé à partir de 0 : la case d'indice i contient soit un triplet (nom de variable, numéro du fils gauche, numéro du fils droit) si le nœud numéro i est un nœud interne, soit un booléen si le nœud numéro i est une feuille. Plus précisément, on définit le type :

```
type noeud =
| Feuille of bool
| Decision of string * int * int
| Vide;;
```

et un arbre de décision est représenté par un tableau de nœuds :

```
type decision = noeud array;;
```

Les nœuds vides permettent de modifier *a posteriori* l'arbre de décision : on déclare un tableau de longueur strictement supérieure au nombre de nœuds et on déclare vides toutes les cases du tableau inutiles. Ceci permet, si on doit modifier la formule, de modifier l'arbre de décision en conséquence.

Dans toute la suite, l'égalité considérée sur les formules est l'égalité sémantique : deux formules sont identifiées dès lors qu'elles ont la même table de vérité.

Exemple 1 :
Si l'on ordonne les nœuds par l'ordre du parcours en profondeur préfixe, l'arbre de décision de la **figure 2** est représenté par

```
let exemple_ad : decision =
        [|Decision("e",1,2);
        Feuille true;
        Decision("a",3,6);
        Decision("r",4,5);
        Feuille true;
        Feuille false;
        Feuille false|];;
```

A.1. Redéfinir la variable `exemple_ad` en numérotant les nœuds dans l'ordre du parcours en largeur.

On pourrait bien sûr utiliser pour cet exemple un tableau de longueur plus grande, dans l'idée de modifier ultérieurement la formule. Les cases suivantes seraient affectées à Vide.

Dans les deux questions suivantes, on veut faire déterminer une décision en fournissant une valuation des variables, sous forme de liste des seules variables qui sont vraies dans l'évaluation.

A.2. Définir une fonction `eval_var : string` → `string list` → `bool` telle que, si `v` est le nom d'une variable et `l` une liste des variables vraies, alors `eval_var v l` retourne le booléen `true` si la variable `v` appartient à `l`, et le booléen `false` sinon.

A.3. Définir une fonction `eval : decision` → `string list` → `bool` telle que, si `a` est un arbre de décision et `l` est une liste des seules variables vraies, alors `eval a l` retourne le booléen correspondant à la décision finale.

B. Réduction d'un arbre de décision à un diagramme de décision

On souhaite compacter la représentation en mémoire des arbres de décision. Si plusieurs sous-arbres sont identiques, on n'a pas envie de les stocker plusieurs fois.

En raisonnant sur la représentation informatique des arbres de décision, on voit assez facilement une façon de procéder : si les arbres de numéros i et j sont identiques, on peut (par exemple) au niveau du parent p de j indiquer comme numéro de fils i au lieu de j et ainsi éliminer j de la représentation.

Ce faisant, on ne représente plus un arbre (car i a maintenant deux parents), mais un graphe orienté : on parle alors de diagrammes de décision. Il existe deux types d'arcs dans les diagrammes de décision suivant que l'arc est suivi dans le cas où la variable est vraie ou dans le cas où la variable est fausse. On notera $p \xrightarrow{b} i$ avec $b \in \{T, F\}$ (pour *vrai*
ou *faux*), selon que l'arc est suivi dans le cas où p est vrai (précédemment : fils gauche) ou dans le cas où p est faux (précédemment : fils droit). Lorsqu'on a $p \xrightarrow{b} i$ on note $i = \text{succ}_b(p)$.

Exemple : l'expression $(z_1 \wedge z_3) \vee (z_2 \wedge z_3)$ admet (entre autres) les diagrammes de décision ci-dessous.

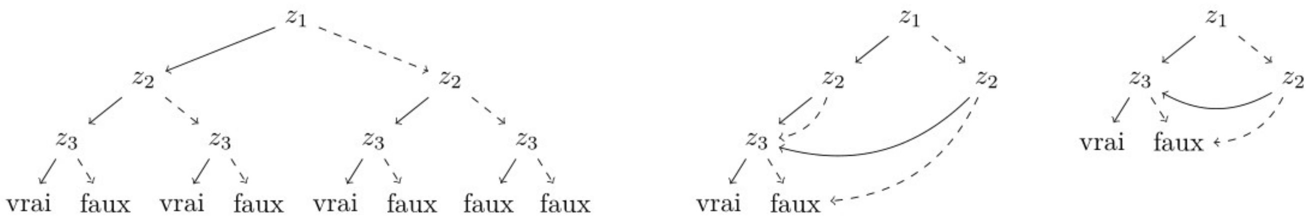

Figure 3 : trois diagrammes de décision associées à la formule $(z_1 \wedge z_3) \vee (z_2 \wedge z_3)$

On notera qu'un diagramme de décision est associé à une formule logique et donc est un graphe assez particulier : en particulier, chaque sommet possède exactement 0 ou 2 arcs sortants.

B.1. Montrer également qu'un tel graphe est nécessairement connexe et acyclique (en tant que graphe orienté).

On souhaite maintenant modifier un arbre de décision en diagramme de décision sans répétition. On va appliquer les deux règles de réécriture suivantes :

— *Élimination :* Si pour un nœud v on a $\text{succ}_F(v) = \text{succ}_T(v) = w$ alors on élimine v et on transforme les arcs de la forme $u \xrightarrow{b} v$ en v.

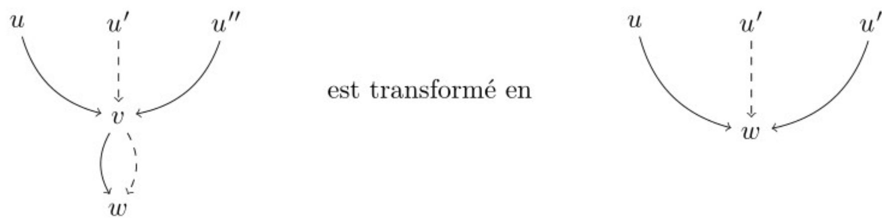

Figure 4 : règle d'élimination.

— *Isomorphisme :* Soit v et w deux nœuds distincts. Si ce sont des feuilles de même valeur ou si ce sont des nœuds internes de même variable et de mêmes transitions, alors on élimine v et on transforme les arcs de la forme $u \xrightarrow{b} v$ en $u \xrightarrow{b} w$.

Figure 5 : règle d'isomorphisme.

B.2. Écrire une fonction `redirige : decision → int → int → unit` prenant comme argument un diagramme et deux indices v et w, ne retournant rien mais ayant pour effet de bord de supprimer le nœud v dans le graphe et de transformer tous les arcs de la forme $u \xrightarrow{b} v$ en $u \xrightarrow{b} w$. Les cases du tableau qui deviennent inoccupées sont remplies avec la valeur Vide.

B.3. Écrire une fonction `trouve_elimination : decision → int` prenant comme argument un diagramme de décision et retournant l'indice d'un nœud pouvant être supprimé par élimination s'il en existe un ; sinon, elle doit retourner -1.

B.4. De même, écrire une fonction `trouve_isomorphisme : decision → int*int`, prenant comme argument un diagramme, et retournant un couple d'indices correspondant à deux nœuds pouvant être simplifiés par isomorphisme s'il en existe un ; sinon, elle doit retourner le couple $(-1, -1)$.

On dit que le diagramme est *sous forme réduite* si aucune des deux règles d'élimination ou d'isomorphisme ne peut s'appliquer.

B.5. Écrire une fonction `reduit : decision` $\to$ `unit`, prenant comme argument un diagramme, et ne retournant rien, qui détecte les deux simplifications possibles, effectue les redirections correspondantes, jusqu'à ce qu'il ne soit possible de faire aucune simplification supplémentaire.

On obtient à ce stade une représentation du diagramme simplifié sous forme d'un tableau dans lequel certaines cases ne sont plus utilisées : elles sont affectées à Vide.

Un diagramme de décision réduit occupe donc moins de place que tout diagramme de décision à partir duquel il a été obtenu. Cependant, pour une même formule logique, un diagramme de décision réduit n'est ni unique, ni même minimal, comme le montre l'exemple de la figure 6 page suivante.

Aucune règle de simplification ne peut être appliquée au diagramme de la figure 6, et pourtant ce diagramme représente la formule $b \wedge c$ qui peut être représentée par des diagrammes plus petits sur lesquels ne figure pas a. Le problème ici est l'ordre d'apparition des variables lorsqu'on parcourt un chemin dans le diagramme, comme on va le voir plus bas.

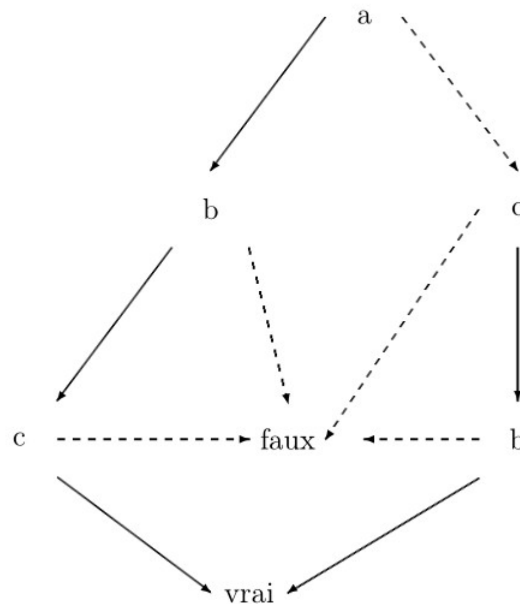

Figure 6 : un diagramme de décision réduit pour la formule b/\c.

C. Satisfiabilité d'une formule donnée par un diagramme de décision

Étant donné un diagramme de décision (réduit ou pas) associé à une formule logique, la satisfiabilité de la formule se traduit par un chemin dans le diagramme conduisant à une feuille `true`.

C.1. Écrire une fonction `satisfiable_profondeur : decision` $\to$ `bool` prenant comme argument un diagramme de décision et retournant le booléen indiquant si la formule correspondante est satisfiable. **Cette fonction devra utiliser comme stratégie de parcours un parcours en profondeur.** On utilisera une liste pour garder en mémoire les indices des nœuds déjà visités.

C.2. Écrire une fonction `satisfiable_largeur : decision` $\to$ `bool` prenant comme argument un diagramme de décision et retournant le booléen indiquant si la formule correspondante est satisfiable. **Cette fonction devra utiliser comme stratégie de parcours un parcours en largeur.** On utilisera une liste pour garder en mémoire les indices des nœuds déjà visités.

D. Construction d'arbres et de diagrammes de décision

Nous nous intéressons maintenant à la construction de diagrammes de décision à partir de formules logiques. Étant données des formules logiques t, e_1 et e_2, on définit un connecteur ternaire $\cdot \to \cdot\cdot, \cdot\cdot\cdot$ par :

$$t \to e_1, e_2 \;=\; (t \wedge e_1) \vee (\neg t \wedge e_2)$$

D.1. Si v est une variable propositionnelle et f_1, f_2 deux formules logiques, dessiner un arbre de décision de $v \to f_1, f_2$ en fonction d'un arbre de décision de f_1 et d'un arbre de décision de f_2.

D.2. Soient v une variable propositionnelle et e une formule logique. Que vaut $v \to e, e$?
À quoi cette égalité correspond-elle ?

On utilise maintenant le connecteur $\cdot \to \cdot\cdot, \cdot\cdot\cdot$ pour construire des **arbres** de décision à partir d'une formule.

D.3. Soient x et y des formules logiques quelconques. Montrer que les formules logiques $x \Rightarrow y$, $\neg x$, $x \vee y$, $x \wedge y$ et $x \Leftrightarrow y$ peuvent s'écrire en utilisant uniquement les constantes 0 (faux), 1 (vrai), l'opérateur $\cdot \to \cdot\cdot, \cdot\cdot\cdot$ défini précédemment et les formules x et y.

D.4. En déduire que toute formule propositionnelle peut s'écrire en utilisant uniquement les constantes 0 (faux), 1 (vrai), l'opérateur $\cdot \to \cdot\cdot, \cdot\cdot\cdot$ et les variables propositionnelles.

D.5. Pour toutes formules a, b, c, d, e, montrer qu'on a $(a \to b, c) \to d, e = a \to (b \to d, e), (c \to d, e)$.

D.6. Pour toutes formules e_1 et e_2, simplifier $0 \to e_1, e_2$ et $1 \to e_1, e_2$.

On déduit de ce qui précède un algorithme de construction d'un arbre de décision à partir d'une formule logique quelconque :

- à l'aide du résultat de la question D.4, on réécrit la formule logique en n'utilisant que les constantes et l'opérateur $\cdot \to \cdot\cdot, \cdot\cdot\cdot$;
- tant qu'il reste à gauche d'une flèche $\to$ une expression non réduite à une constante ou une variable, on la réécrit à l'aide de la question D.5. ;
- tant qu'il reste à gauche d'une flèche une constante, on l'élimine grâce à la question D.6.

Cet algorithme termine car la somme des longueurs des expressions à gauche des flèches diminue strictement à chaque étape. La formule obtenue correspond à un arbre de décision.

D.7. Appliquer cet algorithme à la formule $(a \Rightarrow b) \Rightarrow (a \Rightarrow a)$.

Il suffit alors d'appliquer la fonction `reduit` pour obtenir un diagramme de décision réduit associé à la formule logique. Mais cette méthode de construction d'un arbre de décision ne respecte pas forcément un certain ordre des variables. À cause de cela, le diagramme de décision réduit obtenu peut être inutilement volumineux.

E. Diagrammes de décision ordonnés

Un diagramme de décision est dit **ordonné** si, pour un ordre donné entre les variables $x_1 \prec x_2 \prec \ldots \prec x_n$, alors tout chemin partant de la racine vers les feuilles parcourt les variables dans cet ordre.

Dans cette partie nous proposons une autre méthode de construction que celle vue précédemment, un ordre étant donné *a priori*.

La formule propositionnelle représentée par un diagramme de décision u est notée f^u (on rappelle que l'égalité sur les formules est ici l'égalité sémantique).

Pour une variable v, une formule $expr$ et une formule f, on note $f[v = expr]$ la formule obtenue à partir de f en remplaçant toutes les occurrences de la variable v par la formule $expr$.

E.1. Soit v une variable propositionnelle et f une formule logique. Que vaut $v \to f[v = 1], f[v = 0]$?

On remarque (il n'est pas demandé d'épiloguer sur ce point) que pour un diagramme donné, les deux règles d'élimination et d'isomorphisme préservent l'ordre des variables dans les chemins de la racine à une feuille.

On en déduit un algorithme de construction d'un diagramme de décision réduit ordonné à partir d'une formule logique sur un ensemble ordonné de variables. On construit le diagramme ordonné récursivement : en supposant construits des diagrammes ordonnés d_1 et d_0 de $f[x_1 = 1]$ et de $f[x_1 = 0]$ (considérées comme des formules de $n-1$ variables ordonnées), on forme le l'arbre de décision de racine x_1 qui a pour successeurs les racines de d_1 et d_0. Le cas de base est celui d'une fonction de zéro variables, qu'on peut alors évaluer à une constante booléenne : dans ce cas le diagramme ordonné est une feuille. Enfin, on applique la fonction `reduit` à l'arbre obtenu.

E.2. Appliquer cet algorithme (y compris la phase de réduction) à la formule $(a \Rightarrow b) \Rightarrow (a \Rightarrow a)$ (on choisit l'ordre alphabétique).

E.3. Justifier que pour toute formule logique f de n variables ordonnées $x_1 \prec x_2 \prec \ldots \prec x_n$, il existe un unique diagramme de décision réduit ordonné u tel que $f^u = f$. Attention, on parle bien ici de diagrammes et pas de leur implémentation : l'unicité ne vaut qu'à renumérotation des nœuds près si ce sont les impémentations que l'on compare.

E.4. À l'aide de ce qui précède, comment déterminer, le plus simplement possible :

— si une formule logique est une tautologie ?
— si une formule logique est une satisfiable ?
— si deux formules logiques portant sur le même ensemble de n variables sont égales ?