Formules logiques et BDD

Dans le makefile :

- **Règle "ocaml" :**
  - **ocaml app.ml :**
    Cette règle exécute le fichier **app.ml** en utilisant l'interpréteur OCaml. Lorsque vous exécutez la commande **make ocaml**, cela exécutera votre programme OCaml défini dans **app.ml**. Assurez-vous que le fichier **app.ml** contient le code source approprié pour votre application OCaml.

- **Règle "pdf" :**
  - **dot -Tpdf -o src/pdf/dotDec.pdf src/dot/dotDec.dot :**
    Cette règle utilise l'outil **dot** de Graphviz pour convertir le fichier DOT **src/dot/dotDec.dot** en un fichier PDF **src/pdf/dotDec.pdf. Dot** est un programme qui prend en entrée un fichier DOT et génère un fichier graphique correspondant. L'option **-Tpdf** spécifie le format de sortie comme PDF.
  - **dot -Tpdf -o src/pdf/dotBDD.pdf src/dot/dotBDD.dot :**
    Cette règle est similaire à la précédente, mais elle convertit le fichier DOT **src/dot/dotBDD.dot** en un fichier PDF **src/pdf/dotBDD.pdf**.

- **Règle "clean" :**
  - **rm -rf src/dot/* :**
    Cette règle supprime tous les fichiers présents dans le répertoire **src/dot/** et ses sous-répertoires. L'option **-rf** supprime récursivement les fichiers et les répertoires, y compris les sous-répertoires.
  - **rm -rf src/pdf/* :**
    Cette règle supprime tous les fichiers présents dans le répertoire **src/pdf/** et ses sous-répertoires.

Dans l'app.ml :

- **Fonction getVars** :
  - **Description :** Cette fonction prend une formule logique en entrée et retourne la liste des variables présentes dans la formule, triée par ordre croissant.
  - **Choix et justifications :**
    - Utilisation d'une fonction auxiliaire récursive : La fonction auxiliaire parcourt récursivement la formule et ajoute les variables rencontrées à une liste accumulatrice.
    - Utilisation de **List.sort_uniq** : Pour éliminer les doublons et trier la liste des variables en ordre croissant, nous utilisons la fonction **List.sort_uniq** avec le comparateur **String.compare**.
  - **Type**: tformula -> string list

- **Fonction string_of_var :**
  - **Description :** Cette fonction convertit une variable de type tformula en sa représentation sous forme de chaîne de caractères.
  - **Choix et justifications :**
    - Utilisation d'un motif de correspondance : Nous vérifions si la variable est de type **Var** et retournons la chaîne de caractères correspondante.
    - Gestion d'erreur : Si la variable n'est pas de type **Var**, une exception est levée.
  - **Type:** tformula -> string

- **Fonction evalFormula :**
  - **Description :** Cette fonction évalue une formule logique dans un environnement donné.
  - **Choix et justifications :**
    - Utilisation d'une fonction auxiliaire récursive : La fonction auxiliaire parcourt récursivement la formule et évalue chaque sous-formule en fonction des valeurs des variables dans l'environnement.
    - Utilisation de la fonction **List.assoc** : Pour obtenir la valeur d'une variable dans l'environnement, nous utilisons la fonction **List.assoc** qui recherche la première paire clé-valeur correspondante.
  - **Type:** (string * bool) list -> tformula -> bool

- **Fonction buildDecTree :**
  - **Description :** Cette fonction construit l'arbre de décision d'une formule logique.
  - **Choix et justifications :**
    - Utilisation d'une fonction auxiliaire récursive : La fonction auxiliaire prend en compte l'environnement et une liste de variables, et construit l'arbre de décision en fonction des valeurs possibles des variables.
    - Utilisation de la structure de données récursive **decTree** : L'arbre de décision est représenté par les types **DecLeaf** (feuille) et **DecRoot** (nœud). Chaque nœud contient une variable et deux sous-arbres correspondant aux valeurs Vrai et Faux de la variable.
  - **Type:** tformula -> decTree

- **Fonction buildBdd (ne fonctionne pas correctement) :**
  - **Description :** Cette fonction construit un BDD (Diagramme de Décision Binaire) à partir d'une formule logique.
  - **Choix et justifications :**
    - Utilisation d'une fonction auxiliaire récursive : La fonction auxiliaire prend en compte les variables, la formule et construit le BDD en utilisant une table de nœuds accumulée.
    - Utilisation de la structure de données **bddNode** : Un nœud du BDD est représenté par les types **BddLeaf** (feuille) et **BddNode** (nœud). Chaque nœud contient une variable et deux sous-nœuds correspondant aux valeurs Vrai et Faux de la variable.
  - **Type :** tformula -> bddNode

- **Fonction simplifyBDD (ne fonctionne pas correctement sans buildBdd) :**
  - **Description :** Cette fonction simplifie un BDD en éliminant les nœuds redondants et en fusionnant les nœuds équivalents.
  - **Choix et justifications :**
    - Utilisation d'une fonction auxiliaire récursive : La fonction auxiliaire parcourt récursivement le BDD en utilisant une table de nœuds accumulée et applique les règles de simplification pour éliminer les nœuds redondants.
    - Utilisation de la fonction **bddNode_equal** : Pour détecter les nœuds équivalents, nous utilisons la fonction **bddNode_equal** qui compare les nœuds en fonction de leur variable et de leurs sous-nœuds.
  - **Type :** bddNode -> bddNode

- **Fonction isTautology :**
  - **Description :** Cette fonction vérifie si une formule logique est une tautologie, c'est-à-dire si elle est vraie pour toutes les assignations de variables.
  - **Choix et justifications :**
    - Utilisation de la fonction **buildBdd** : Nous construisons le BDD correspondant à la formule logique en utilisant la fonction **buildBdd**.
    - Vérification de la présence d'une feuille True : Si le BDD contient une feuille True, cela signifie que la formule est une tautologie.
  - **Type :** tformula -> bool

- **Fonction areEquivalent :**
  - **Description :** Cette fonction vérifie si deux formules logiques sont équivalentes, c'est-à-dire si elles ont la même valeur de vérité pour toutes les assignations de variables.
  - **Choix et justifications :**
    - Utilisation de la fonction **buildBdd** : Nous construisons les BDDs correspondants aux deux formules logiques en utilisant la fonction **buildBdd**.
    - Comparaison des BDDs : Si les deux BDDs sont égaux, cela signifie que les formules sont équivalentes.
  - **Type :** tformula -> tformula -> bool

- **Fonction dotBDD (ne fonctionne pas correctement) :**
  - **Description :** Cette fonction génère une représentation graphique au format DOT d'un BDD.
  - **Choix et justifications :**
    - Utilisation d'une fonction auxiliaire récursive : La fonction auxiliaire parcourt récursivement le BDD en générant les instructions DOT correspondantes pour chaque nœud.
    - Utilisation de la notation DOT : La notation DOT est un langage de description graphique largement utilisé pour représenter des graphes.
  - **Type :** bddNode -> string

- **Fonction dotDec :**
  - **Description :** Cette fonction génère une représentation graphique au format DOT d'un arbre de décision.
  - **Choix et justifications :**
    - Utilisation d'une fonction auxiliaire récursive : La fonction auxiliaire parcourt récursivement l'arbre de décision en générant les instructions DOT correspondantes pour chaque nœud.
    - Utilisation de la notation DOT : Nous utilisons la notation DOT pour représenter l'arbre de décision de manière graphique.
  - **Type :** decTree -> string