Cycle ingénieur - 2ème année MI

Programmation fonctionnelle

Projet *Formules logiques*

2023-2024

Consignes

Consignes générales

Objectif du projet

- Manipuler des formules logiques (substition, évaluation, normalisation)

Consignes générales

- Projet effectué par groupes de **4 à 5 étudiants**
- Date limite de rendu (sur TEAMS) : **3 décembre 2023, 23h59**
- Soutenance : semaine du **4 au 8 décembre 2023**

Nature du rendu

- *Fourni* : Trois modules (`Formula`, `Literal` et `NormalForm`)
- **À faire** : implémenter les éléments manquants dans chaque module
- **Rendu** : archive contenant les trois modules et rapport court

Consignes spécifiques

Modification du code existant

- Modification **INTERDITE** :
  - de la signature des modules
  - de la signature des valeurs et fonctions à implémenter
  - des (quelques) éléments déjà implémentés
- Ajout **INTERDIT** de paquetages supplémentaires
- Ajout *autorisé* de nouveaux éléments intermédiaires pour implémenter les éléments demandés

Tests des valeurs et méthodes à implémenter

- Des tests seront effectués sur votre code. **Ces tests participent à la note finale du projet.**
- Pour fonctionner, **ces tests supposent que les consignes précédentes ont été respectées**. *Dans le cas contraire, les tests ne fonctionneront pas.*

Consignes spécifiques

Respect des règles de programmation fonctionnelle

- Utilisation de variables **interdite**
- Utilisation des boucles (conditionnelles ou non) **interdite**

Quelques conseils

- **Réduire le recours à la récursivité** en utilisant les fonctions du paquetage standard
- sinon **favoriser la récursivité terminale**

Rapport

Description

- Description succincte de la solution mise en place
- Difficultés rencontrées **et leur résolution**

Analyse

- Analyse de complexité

Soutenance (semaine du 4/12 au 8/12)

Organisation

- Durée totale : *20 minutes*
  - Temps de présentation : **entre 10 et 15 min**

    Contrainte de temps à respecter absolument
  - Reste du temps consacré à nos questions
- **Tous les membres du groupe doivent présenter.**

Attendu

- Présenter la solution implémentée
- **Présenter des jeux de test pour valider l'implémentation**

Présentation du projet

Présentation générale

TD n°3, exercice 5 : formules avec une variable logique

- Évaluation
- Équivalence
- Simplification

Généralisation à plusieurs variables logiques

- Évaluation
- Simplification
- Forme normale conjonctive
- Principe de résolution de ROBINSON

Module Formula

Type `Formula` à définir

- Formule logique avec un nombre *quelconque* de variables logiques

À implémenter

- Constructeurs de base, opérations usuelles
- Recherche d'une variable logique, ensemble des variables
- Évaluation, équivalence logique, simplification

Classes de types à associer

- Instanciation de la classe Show
- **Dérivation de la classe Eq à rajouter** *(décommenter la ligne)*

```
data Formula = ...
    deriving Eq
```

Module `Literal`

Type `Literal` à définir

- Littéral comme partie atomique d'une forme normale

À implémenter

- Constructeur de base
- Négation
- Conversion en formule logique

Classes de types à associer

- Instanciation de la classe Show
- **Dérivation des classes `Eq` et `Ord` à rajouter** *(décommenter la ligne)*

```
data Literal = ...
    deriving (Eq, Ord)
```

Module NormalForm

Type `CNF` (*Conjunctive Normal Form*) fourni

- Utilise `Data.Set` *(dépendance déjà mise en place)* :

`{{A1, A2},{B},{C1, C2, C3}}` ≡ (A1 ∨ A2 ∨ A3) ∧ B ∧ (C1 ∨ C2 ∨ C3)

À implémenter

- Taille (en nombre de littéraux)
- Conversion de et vers une formule logique
- Application du principe de résolution de ROBINSON

Classes de types à associer

- Instanciation de la classe Show
- **Dérivation de la classe Eq à rajouter** *(décommenter la ligne)*

```
newtype CNF = CNF (Set (Set Literal))
    deriving Eq
```

Compilation « à vide »

Le code fourni compile déjà

- Les types à définir sont simplement déclarés… comme des types.
- Tous les autres éléments à implémenter ont la valeur `undefined`.

Implémentation des types

1. Remplacer la ligne : `data ... :: Type`
   par votre implémentation : `data ... = ...`
2. Décommenter la dérivation des classes de type : `deriving...`
3. Enlever les lignes devenues inutiles :
   - `{-#LANGUAGE KindSignatures #-}` *(première ligne du fichier)*
   - `import Data.Kind`