Rapport de projet

Implémentation d'un tableur de calcul ExceML

Auteur : CAO Gabriel
Numéro étudiant : 3679695

Sous la supervision de :
Rémy El Sibaie

Département de licence d'informatique
Sorbonne-Universités
Mars 2020

Table des matières

Base du projet

Cette première partie du projet est constitué de l'implémentation de la logique et du fonctionnement interne du tableur qui sera développé en OCaml.

La grille du tableur sera modélisé par une matrice de cases.

Les cases de notre tableur seront des expressions pouvant contenir diverses valeurs.

Une expression est :

- *Vide* : absence de valeur
- *Chaîne* : une chaîne de caractères constante
- *Entier* : une constante entière
- *Flottant* : une constante flottante
- *Case* : une référence vers une autre case
- *Opération binaire* : une fonction d'arité 1 telle que la valeur absolue d'un entier
- *Opération unaire* : une fonction d'arité 2 telle que l'addition
- *Opération de réduction* : une fonction d'arité n telle que une somme globale, une moyenne

Une expression est évaluée et renvoie un résultat.

Si une expression incorrecte est détectée alors une erreur est renvoyée.

Les erreurs possibles sont :

- *Mauvais indice* : lorsque l'utilisateur effectue une référence vers une case inexistante, par exemple, si le format de la grille est 10x10 et que l'utilisateur fait une référence vers la case (100,100) alors cette erreur est renvoyée.
- *Mauvaise référence* : lorsque l'utilisateur effectue une référence vers une case contenant une erreur.
- *Cycle* : lorsque le calcul souhaité est impossible à réaliser car il aboutit à un cycle.
- *Mauvais Argument* : lorsqu'une fonction est mal utilisée par l'utilisateur, par exemple, la somme d'une chaîne de caractères et d'un entier.

Choix techniques

Types implémentés de base

- *grille* : modélisation de la grille du tableur.
- *expr* : expression présente dans les cases de la grille.
- *resultat* : résultat de l'évaluation d'une expression.
- *erreur* : erreur renvoyé lorsqu'une expression est non valide.

Fonctions implémentés de base

- **cree_grille** : fonction de signature int → int → expr construisant une grille de hauteur et de largeur spécifiés et dont les cases sont initialisés avec l'expression Vide.

  => Choix d'implémentation : utilisation de la fonction make_matrix de la librairie standard Array

  =>Explication : Ce choix permet de rendre le code plus concis et plus explicite

- **expr_to_string** : fonction de signature expr → string permettant d'obtenir la chaîne de caractères associée à une expression.

  => Choix d'implémentation: utilisation d'un filtrage par motif match … with

  =>Explication : Ce choix permet d'avoir un code plus lisible et plus simple à debugger

- **affiche_grille** : fonction de signature grille → unit permettant d'afficher la grille entière.

  =>Choix d'implémentation : pour afficher une expression de la grille, la fonction est utilisée expr_to_string pour obtenir une chaîne de caractères et la fonction printf de la librairie Format est utilisée pour afficher sur la sortie standard une chaîne de caractères. Pour parcourir la grille, la fonction iter de la librairie Array est utilisée.

  =>Explication : la fonction printf permet d'améliorer la visibilité de l'affichage. La fonction iter rend le code plus concis car elle remplace des filtrage par motifs successifs ou for boucles for imbriqués si cette fonction devait être programmé en style impératif.

- **Cycle** : fonction de signature grille → expr → bool permettant la détection de dépendances cycliques dans la grille spécifiée lors de l'évaluation de l'expression spécifiée.

  =>Choix d'implémentation : Le parcours effectué par l'évaluation de l'expression doit être sauvegarder pour pouvoir détecter une dépendance cyclique. La fonction utilise une référence vers une liste pour modéliser cette ‘‘mémoire’’. Si un cycle existe mais que une erreur d'évaluation est détectée alors la priorité revient à l'erreur évaluée qui est alors remontée.

- **eval_expr** : fonction de signature expr → resultat permettant d'évaluer l'expression spécifié.

  => Choix d'implémentation : un motif de filtrage est utilisé pour analyser chaque cas. Lorsque l'évaluation d'une expression évalue implicitement une référence vers une autre case, alors une détection de cycle et une détection d'indice doivent être employées.

- **affiche_resultat**:miroir de la fonction affiche_expr pour un resultat.
- **affiche_grille_resultat** : miroir de la fonction affiche_grille pour une grille de resultat.
- **myabs** : fonction de signature expr → expr prenant en argument une expression v et construisant une expression Unaire <op> qui retourne la valeur absolue de v.

  => Choix d'implémentation : cette fonction utilisera la fonction auxiliaire absolu dans son champ app1
- **absolu** :fonction de signature resultat → resultat renvoyant la valeur absolue d'un flottant ou d'un entier.

  =>Choix d'implémentation : cette fonction utilise un motif de filtrage et renvoie une erreur signalant l'entrée d'argument non valide si l'entrée par l'utilisateur est différente d'un entier ou d'un flottant.
- **add** : fonction de signature expr → expr → expr prenant en argument deux expressions v1 et v2 et construisant une expression Binaire <binop> qui retourne la somme de v1 et v2.

  => Choix d'implémentation : cette fonction utilisera la fonction auxiliaire addition dans son champ app2
- **addition**: fonction de signature resultat → resultat → resultat renvoyant la somme de deux flottant ou de deux entiers.

  =>Choix d'implémentation : cette fonction utilise un motif de filtrage et renvoie une erreur signalant l'entrée d'argument non valide si l'entrée par l'utilisateur est différente d'un couple d'entiers ou d'un couple de flottants.
- **somme** : fonction de signature (int * int) → (int * int) → expr prenant en argument une case de début et une case de fin soit deux couple d'entiers et construisant une expression Reduction <redop> qui retourne la somme des cases du rectangle formée par les deux cases spécifiées.

  => Choix d'implémentation : cette fonction utilisera la fonction auxiliaire addition dans son champ app

Analyse d’algorithmes

- Algorithme de détection de cycle :
- En cours d’implémentation

Améliorations

- Implémentation d’un mécanisme de mémoïsation lors de l’évaluation d’une grille.
- Implémentation de nouvelles fonctionnalités
  - Opérations unaires
    - Inverse d’un entier ou d’un flottant.
    - Opposé d’un entier ou d’un flottant.
  - Opérations binaires
    - Soustraction
    - Multiplication
    - Division
    - Maximum
    - Minimum
    - Moyenne
    - Puissance
  - Opérations de réduction
    - Produit
    - Maximum
    - Minimum