Comptes-rendus

Romain PRETET[1] and Camille VEDANI[2]

1 Université III Paul Sabatier, France, romain.pretet@univ-tlse3.fr

2 Université III Paul Sabatier, France, camille.vedani@univ-tlse3.fr

Résumé

Compte rendu des réunions du TER 2025. Lien du projet :
https://fr.overleaf.com/3768471444ktsnghtzntdj#0fe84e

1 Le 15/05/25

Classe ELM dans Otawa

- **Localisation** : Dans les documents sur le site.
- **Fichiers** : `.elf` (exécutables pour différentes machines, format binaire).

Outils Graphviz

- Intégration directe avec Otawa, utilisé dans XDD.
- Utilisation recommandée avec `xdot` (nécessite installation).
- À éviter pour les gros fichiers.

Fonctionnement d'Otawa

- **Classe `First`** : Gère la classe application.
- **Fonction `work`** : Exécutée par le programme lors de l'utilisation.
- **Objets** : Chaque objet est une liste de propriétés, permettant d'accrocher des propriétés pour des analyses successives.
- **Parcours de CFG** : Utilisation de marqueurs pour éviter les boucles infinies.
  - Recommandation : Bien refaire la version 2.4 et ajouter le marquage dans le programme `.cpp`.
- **CFG Collection** : Otawa fournit les CFG, pas besoin de les créer manuellement.
  - Un programme est une collection de CFG.

Objectifs

- Considérer les XDD comme des fonctions.

Hashtable et Multithreading

- **Hashtable** : Utilisable en multithread.
- **XDD** : Stocké dans une hashtable partagée.
- **Politique de Partage** : Nécessité de développer une politique de partage pour éviter les conflits.
  - **NodeManager** : Liste des XDD retenus.
  - **OpsManager** : Enregistre les opérations déjà calculées.

Recommandations Techniques

- Reprendre la classe `HashMap` ou une surclasse gérant les accès parallèles.
- Mettre à jour `Ops` et `Node` pour assurer le bon fonctionnement.
- Commande à mettre dans bashrc : `export PATH=$PATH::$HOME/otawa/bin`

2 Le 26/05

2.1 Méthodologie de calcul des temps

Nous calculons les temps des XDDs en utilisant un pointeur de temps stocké dans le bloc de base. Notre implémentation intervient après la matrice et l'étape de temporisation (*timing step*).

2.1.1 Gestion des XDDs

Les XDDs sont construits à partir du standard `XDD Manager`, qui comprend :
- Un **Node Manager**
- Un **Ops Manager**

Nous avons modifié ces deux composants (`nodes` et `ops`) en nous basant sur les deux opérations déjà réalisées précédemment.

2.1.2 Implémentation de la Hashmap

La structure de hachage est implémentée dans la classe `ELM` qui fournit :
- Un itérateur (à ne pas modifier)
- Les principales fonctionnalités :
  - `ADD` : pour l'insertion
  - `GET` (qui peut appeler `FETCH`) : récupère une valeur dans le tableau
  - `ASK_KEY` : vérifie la présence d'une clé

2.1.3 Mécanisme de parallélisation

- Encapsulation de la hashmap avec des `mutex` pour la synchronisation
- Implémentation sous forme de table de hachage ouverte avec résolution des collisions par listes chaînées

Le professeur fournira un module pour le calcul de temps orienté parallélisme que nous testerons avec notre hashmap parallélisée.

2.1.4 Gestion des threads

La documentation `ELM (autodoc)` décrit :
— La classe `Mutex`
— Le système de threads

Le professeur utilisera un `JobScheduler` comprenant :
— Un pool de threads
— Un `Job Producer` qui distribue les tâches

Nous évaluerons si des compléments sont nécessaires concernant :
— L'implémentation des threads
— Les mécanismes de mutex
pour une adaptation sous Linux.