Projet du cours « Compilation »
Jalon 8 : Compilation efficace de l'analyse par motifs

version numéro `Fri 28 Feb 2020 09:42:28 AM CET`

1 Présentation de l'optimisation

La traduction naïve de l'analyse par motifs de Hopix vers Hobix utilise une séquence d'expression conditionnelle. Cette approche a un défaut de taille : elle produit généralement du code effectuant des tests redondants. Comme expliqué dans l'article de Luc Maranget :

`http://moscova.inria.fr/~maranget/papers/ml05e-maranget.pdf`

il est possible de compiler les analyses par motifs vers des arbres de décision. Ces arbres factorisent des tests partagés par plusieurs branches. Dans cet article, on apprend aussi que tous les arbres de décision ne se valent pas et qu'il est possible d'effectuer moins de tests en pire cas en choisissant un arbre de décision plutôt qu'un autre.

Pour réaliser ce jalon, vous allez essentiellement modifier le module `PatternMatchingCompiler`.

2 Étape 1 : Définir les opérations sur les matrices d'analyse

La passe de compilation nécessite un certain nombre d'opérations sur les matrices. L'objectif de cette première étape est d'implémenter ces opérations.

1. Observez le type `matrix`, quelles sont différences entre sa définition et la définition de l'article ?
2. Implémentez les fonctions `head_constructors`, `specialize`, `default`, `split`, `swap_columns` et `swap_occurrences`.
3. Testez ces fonctions intensivement en insérant des tests unitaires dans le code.

3 Étape 2 : Compiler les matrices vers des arbres de décision

La passe de compilation est définie par induction sur le nombre de ligne de la matrice. Le cas de base est très simple. Le cas de récurrence a deux sous-cas en fonction de la nature de la première ligne.

Remarquez que l'article suppose qu'il n'y a pas d'annotations de type dans les motifs.

— Ecrivez une fonction qui supprime toutes les annotations de type dans les motifs.

— Ecrivez une fonction permettant de déterminer si la matrice est dans le premier ou le second sous-cas de la récurrence. Dans ce second sous-cas, on renverra le premier i tel que la colonne numéro i contient un motif qui n'est pas universel.

— Complétez la fonction `decision_tree_of_matrix`.

4 Étape 3 : Production du code Hobix

La dernière étape consiste à produit le code Hobix à partir de l'arbre de décision. Remarquez que le `switch` de Hobix ne fonctionne qu'avec de petits entiers.

1. Assurez-vous que votre passe de traduction de Hopix vers Hobix utilise le module `PatternMatchingCompiler` lorsque l'option `--fast-match` est activée.
2. Quels nœuds de l'arbre vont pouvoir être traduits vers des `switch` de Hobix ?
3. Complétez la fonction `compile_decision_tree`.
4. Optionnellement : Trouvez un moyen d'éviter la duplication du code des branches quand les motifs contiennent des sous-motifs disjonctifs.

5 Travail à effectuer

La huitième partie du projet est la conception et la réalisation de la traduction des programmes Hopix en programmes Hobix.

Le projet est à rendre **avant le** :

20 mars 2020 à 23h59

Pour finir, vous devez vous assurer des points suivants :

- Le projet contenu dans cette archive **doit compiler**.
- Vous devez **être les auteurs** de ce projet.
- Il doit être rendu **à temps**.

Si l'un de ces points n'est pas respecté, la note de 0 vous sera affectée.

6 Log