Projet du cours « Compilation »
Jalon 7 : Compilation de Hopix vers Hobix

version numéro `Wed 20 Feb 2019 07:54:32 AM CET`

1 Présentation de Hopix

Reportez vous au jalon 2 pour vous remémorer la syntaxe de Hopix.
Remarquez les différences entre Hopix et Hobix :

- En Hobix, il y a une classe de valeurs pour les blocs alloués, le tas mais plus aucune donnée structurée (types sommes ou enregistrement) et il n'y a plus de références, tandis qu'en Hopix, ces derniers sont présents, mais pas les blocs.
- En Hobix, on peut seulement faire des tests "simples" sous la forme d'expression conditionnelle tandis qu'Hopix est muni d'un filtrage par motifs (*pattern-matching*).
- En Hobix, l'unique structure de boucle est la boucle **while** tandis qu'en Hopix, il y a aussi une boucle **for**.

Ces différences doivent donc être prises en charge par la passe de compilation de Hopix vers Hobix dont il est question dans ce jalon. Pour vous aider à le réaliser, on vous guide en décomposant l'écriture de cette passe en plusieurs étapes. Pour le moment, comme pour les précédents jalons, nous allons considérer seulement des programmes Hopix mettant en jeu des litéraux entiers.

Cette passe de compilation peut se réaliser en quatre étapes : on s'intéressera successivement à la compilation des références, des enregistrements et enfin des types sommes. Pour finir, on traduira les boucles **for**.

1.1 Étape 1 : Traduction des références

Pour le moment, les références vont être vues comme des blocs de taille 1. L'allocation, la lecture et l'écriture dans une référence vont donc simplement être implémentées par l'allocation, la lecture et l'écriture dans des blocs.

1. Complétez les cas `Ref`, `Read` et `Assign` de la fonction `HopixToHobix.expression`.

1.2 Étape 2 : Traduction des enregistrements

Pour traduire la construction d'enregistrements et la lecture de champs dans des enregistrement, il faut se donner une stratégie de représentation des enregistrements en blocs. C'est l'environnement de compilation qui va contenir l'association entre une étiquette d'un champ et sa position dans le bloc. Il n'y a pas de difficulté particulière : il faut seulement s'assurer que tous les blocs qui contiennent un certain champ x ont toujours ce champ x à une position fixée à la compilation.

1. Quelle va être votre stratégie de représentation des enregistrements ?
2. Complétez le cas des définitions de types d'enregistrement dans la fonction `HopixToHobix.type definition` associe une position à chaque étiquette de champ qui soit cohérente avec votre stratégie de représentation des enregistrements.

1.3 Étape 3 : Traduction des types sommes

Pour traduire la construction d'une valeur d'un type somme, il faut construire un bloc dans le premier élément permet de caractériser le constructeur de données utilisé pour construire cette valeur. Comme pour les enregistrements, il faut interpréter chaque définition de type somme pour stocker dans l'environnement un entier associé à chaque constructeur. Dès lors, la traduction de l'application d'un constructeur de données est similaire à celle des enregistrements (sauf que l'on fait précéder les valeurs par le numéro du constructeur).

La difficulté réside dans la compilation de l'analyse de motifs. Comme nous l'avons vu en cours, il faut commencer par supprimer les motifs disjonctifs et ensuite compiler une analyse comme une séquence d'expressions conditionnelles qui testent la réussite ou l'échec de la confrontation entre la valeur analysée et le motif de la branche courante. Si le test réussit, il faut exécuter le corps de la branche et sinon, passez à la branche suivante.

1. Implémentez la fonction `HobixToHopix.pattern` en supposant que le pattern pris en argument ne contient aucune disjonction.
2. Implémentez le cas `Case` de la fonction `HobixToHopix.expression`.
3. Implémentez la fonction `HobixToHopix.expand_or_patterns` qui supprime toutes les occurrences de `POr` dans l'ensemble des patterns utilisées par les branches prises en argument. Cette fonction produit une nouvelle liste de branches.

1.4 Étape 4 : Traduction des boucles for

1. Traduisez les boucles `for` en boucles `while`. Conseil : écrivez une fonction qui effectue cette traduction de Hopix vers Hopix, c'est plus facile.

2 Travail à effectuer

La septième 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** :

5 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.

3 Log