Projet du cours « Compilation »
Jalon 5 : Compilation de Fopix vers Retrolix

version numéro Wed 13 Feb 2019 02 :13 :56 PM CET

1 Présentation de Fopix

La syntaxe de Fopix est définie comme suit :

			Programmes
p	::=	$\overline{d}$	*Liste de définitions*
			Définitions
d	::=	**val** $x = e$	*Variables globales*
	|	**def** $f(\overline{x}) = e$	*Fonctions*
	|	**external** f	*Symboles externes*
e	::=	l	*Litéraux*
	|	x	*Variable*
	|	**val** $x = e$ **;** e **end**	*Définition locale*
	|	$f(\overline{e})$	*Appel à une fonction connue*
	|	**call** e **with** $(\overline{e})$	*Appel à une fonction inconnue*
	|	**if** e **then** e **else** e **end**	*Expression conditionnelle*
	|	**while** e **do** e **end**	*Boucle*

Cette grammaire correspond à un sous-ensemble de la syntaxe du module `FopixAST`. Les constructions ignorées pour le moment seront prises en charge plus tard dans le semestre.

Remarquez les différences entre Fopix et Retrolix :

— Fopix ne fait pas référence explicitement aux registres x86-64 contrairement à Retrolix. Les conventions d'appel de fonction ne sont donc pas visibles en Fopix.

— Une expression de Fopix peut contenir plusieurs calculs élémentaires tandis que les instructions de Retrolix sont des opérations élémentaires.

— Une fonction Retrolix déclare les variables locales qu'elle va utiliser tandis qu'en Fopix, il n'y a pas de notion de variables locales.

— Une définition (locale) en Fopix associe un nom à une valeur. Cette construction n'est pas présente en Retrolix.

Ces différences doivent donc être prises en charge par la passe de compilation de Fopix vers Retrolix 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 le jalon 4, nous allons considérer seulement des programmes Fopix mettant en jeu des litéraux entiers.

1.1 Étape 1 : Comprendre Fopix

Pour bien comprendre Fopix, commencez par étudier son analyseur syntaxique (pour comprendre sa syntaxe concrète) et son interpréteur (pour comprendre sa sémantique) en écrivant les programmes suivants dans ce langage :

1. Le programme dont le code source est :

```
val y = 5

val main =
  val y = 3;
```

```
val x = (val y = 4 ; 2);
val z = y;
z + x + y
```

2. Un programme définissant une variable `x` résultat du calcul de l'expression arithmétique $37 * 2 + 3 * 6 - 1$ et une variable `y` résultat du calcul de `x`/2.
3. Un programme qui calcule la somme des 100 premiers entiers à l'aide d'une boucle `while` et qui définit la variable `res` à 0 si ce nombre est plus grand que 1000 ou 1 dans le cas contraire.
4. Un programme qui appelle la fonction `print_int` avec l'argument `37`.
5. Un programme qui définit la fonction `fact` qui attend un entier `n` et calcule $n!$ itérativement. Il définit aussi une variable `res` résultat de l'appel de fonction `fact (5)`.
6. Un programme qui définit la fonction `fact` qui attend un entier `n` et calcule $n!$ récursivement. Il définit aussi une variable `res` résultat de l'appel de fonction `fact (5)`.
7. Un programme qui définit une fonction `add_eight_int` qui attend 8 entiers et renvoie leur somme. Il définit aussi une variable `res` résultat de l'appel de fonction `add_eight_int(1, 2, 3, 4, 5, 6, 7, 8)`.

Vous pouvez tester l'interprétation de vos programmes à l'aide de la commande :

```
flap -s fopix -d true -r true -VV true your-program.fopix
```

1.2 Étape 2 : Simulation des définitions locales

Les variables locales de Retrolix permettent de simuler les définitions locales de Fopix en les remplaçant par des affectations. L'idée est de traduire l'expression :

```
val x = 1 ; ...
```

de Fopix en l'instruction suivante de Retrolix :

```
x <- load 1
```

Seulement, comme le montre le programme (1), cette traduction est trop naïve dans le cas où le même identificateur est réutilisé plusieurs fois en Fopix.

1. Écrivez le code Retrolix pour le programme (1) obtenu en appliquant la règle de traduction ci-dessus. Pourquoi est-ce une traduction incorrecte du programme (1) ?
2. Proposez un prétraitement du programme Fopix permettant d'appliquer cette règle de traduction correctement.
3. Écrivez le code Retrolix obtenu en appliquant la règle de traduction sur le programme Fopix issu du prétraitement que vous avez proposé. Pourquoi est-ce que cette traduction est correcte désormais ?
4. Implémentez votre prétraitement en complétant la fonction `preprocess` du module `FopixToRetrolix`.

1.3 Étape 3 : Décomposition des expressions

Le programme (2) effectue un calcul arithmétique composé de plusieurs opérations élémentaires. Ce calcul doit donc être traduit par plusieurs instructions Retrolix.

Pour décomposer élégamment la traduction des expressions complexes, on a paramétré la fonction `expression` du module `FopixToRetrolix` par une variable `out` représentant la *lvalue* dans laquelle doit être stockée le résultat du sous-calcul en cours de traduction. Ainsi, pour traduire l'expression $1+2*x$, on commence par introduire une variable fraîche y et à rappeler la traduction récursivement en lui passant y et l'expression $2*x$. Comme cette opération de multiplication est atomique, cet appel retourne immédiatement une séquence d'instructions composée d'une unique instruction Retrolix `y <- mul 2, x`. Il suffit alors de faire suivre cette instruction par l'instruction `out <- 1 + y` pour obtenir la série d'instructions simulant les calculs de l'expression $1 + 2 * x$.

1. Observez le code du cas `S.FunCall (S.FunId f, es) when is_binop f` de la fonction `expression` du module `FopixToRetrolix` et associez le code aux explications du paragraphe précédent.
2. En déduire l'implémentation des cas du `IfThenElse` et du `While`.
3. Vérifiez que le programme (3) est correctement compilé par votre traduction.

1.4 Étape 4 : Les conventions d'appels du côté de l'appelant

Les registres x86-64 ne sont pas apparents en Fopix alors qu'ils le sont en Retrolix. C'est aussi en Retrolix que les conventions d'appel des fonctions sont implémentées.

Cette avant-dernière étape se concentre sur l'implémentation des conventions d'appel du côté de l'appelant. En d'autres termes, il s'agit d'implémenter la traduction des expressions de type `FunCall` et `UnknownFunCall`.

On rappelle que l'appelant d'une fonction doit :
— Passer les six premiers arguments dans les registres `%rdi` à `%r9` (voir cours précédent).
— Passer le septième argument et les suivants sur la pile.
— Sauvegarder les registres "caller-save" avant l'appel et les restaurer après l'appel.
— Récupérer le résultat de l'appel de fonction dans `rax`.

1. Introduisez une fonction pour chacun des points cités ci-dessus et implémentez-les.
2. Complétez les cas `FunCall` et `UnknownFunCall` de la fonction `expression`.
3. Vérifiez que le programme (4) est maintenant correctement traduit par votre compilateur.

1.5 Étape 5 : Les conventions d'appels du côté de l'appelé

Pour terminer cette passe de traduction, il faut maintenant s'assurer qu'une fonction appelée implémente correctement les conventions d'appel. On rappelle que :

1. La fonction doit récupérer ses six premiers arguments dans des registres.
2. La fonction doit sauvegarder les registres "callee-save" avant de faire toute autre chose et doit les restaurer avant de rendre la main à son appelant.

1. Rappelez-vous de la syntaxe des définitions de fonction en Retrolix. À quoi correspondent les arguments formels ?
2. Comment traduire simplement la récupération des quatre premiers arguments de façon à se ramener à leurs nommages initiaux dans la suite du code ?
3. Quand peut-on calculer la liste des variables locales à déclarer au début de la fonction ?
4. Complétez le cas `S.DefineFunction (S.FunId f, xs, e)` de la fonction `declaration`.
5. Vérifiez que votre compilateur traduit correctement les programmes (5), (6) et (7).

2 Travail à effectuer

La cinquième partie du projet est la conception et la réalisation de la traduction des programmes Fopix en programmes Retrolix.

Le projet est à rendre **avant le** :

1er mars 2019 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